NAR-2090 Series
Communications Appliance

User's Manual

Revision: 01.0

Portwell Inc.
3F, No. 92, Sec. 1, Nei-Hu Rd., Taipei 114, Taiwan, R.O.C.
Headquarter: +886-2-2799-2020 FAX: +886-2-2799-1010
http://www.portwell.com.tw
E M A I L :
I N F O @ M A I L . P O R T W E L L . C O M . T W

ISO 9001

Table of Contents

Chapter 1 Introduction 2

1.1 About This Manual 2

1.2 Manual Organization 2

1.3 Technical Support Information 2

Chapter 2 Get Started 3

2.1 Included Hardware 3

2.2 Before You Begin….. 3

2.3 The Chassis ……4

2.4 Open the Chassis. 4

2.5 Install or Remove a SODIMM 4

2.7 Install Compact Flash 6

2.8 Install 3.5" Hard disk 7

2.9 Add riser card into system 9

2.10 Default Reset cable & Status cable 10

2.11 Product Specifications 11

2.12 Hardware Configuration Setting 12

2.13 Use a Client Computer 18

2.15 Reset to Default Sample Code information 25

Chapter 3 Operation Guide 51

3.1 Brief Guide of PPAP-2020 51

3.2 System Architecture 52

Chapter 1 Introduction

1.1 About This Manual

This manual describes all required information for setting up and using the NAR-2090 All mentioned below applies to the whole system, unless specially stated.

NAR-2090 provides the essential components for delivering optimal performance and functionality in the value communications appliance market segment. This manual should familiarize you with NAR-2090 operations and functions. NAR-2090 family has one, two or five on-board Ethernet ports to serve communication appliances, such as Firewall, which needs more Ethernet ports to connect external network (internet), demilitarized zone and internal network.

NAR-2090 features:

- Versatile networking and I/O capabilities: 1, 4 or 5 Ethernet ports
- One COM ports
- One miniPCI slot
- Onboard 256MB RAM, Up to 512 Mbytes or 1Gbytes of DDR2 memory

1.2 Manual Organization

The manual describes how to configure your NAR-2090 system to meet various operating requirements. It is divided into three chapters, with each chapter addressing a basic concept and operation of this whole system.

Chapter 1: Introduction. This section briefly talks about how this document is organized. It includes some guidelines for users who do not want to read through everything, but still helps you find what you need.

Chapter 2: Hardware Configuration Setting and Installation. This chapter shows how the hardware is put together, including detailed information. It shows the definitions and locations of Jumpers and Connectors that you can easily configure your system. Descriptions on how to properly mount the main memory are also included to help you get a safe installation. Reading this chapter will teach you how to set up NAR-2090.

Chapter 3: Operation Information. This section gives you illustrations and more information on the system architecture and how its performance can be maximized.

Any updates to this manual, technical clarification and answers to frequently asked questions would be posted on the web site: ***http://isc.portwell.com.tw***

1.3 Technical Support Information

Users may find helpful tips or related information on Portwell's web site: ***http://www.portwell.com.tw.*** A direct contact to Portwell's technical person is also available. For further support, users may also contact Portwell's headquarter in Taipei or your local distributors.

Taipei Office Phone Number: +886-2-27992020

Chapter 2 Get Started

This section describes how the hardware installation and system settings should be done.

2.1 Included Hardware

The following hardware is included in your kit:

- PPAP-2020 Communication Appliance System Board
- One null serial port cable

2.2 Before You Begin

To prevent damage to any system board, it is important to handle it with care. The following measures are generally sufficient to protect your equipment from static electricity discharge:

When handling the board, use a grounded wrist strap designed for static discharge elimination and touch a grounded metal object before removing the board from the antistatic bag. Handle the board by its edges only; do not touch its components, peripheral chips, memory modules or gold contacts.

When handling memory modules, avoid touching their pins or golden edge fingers. Put the value communications appliance system board and peripherals back into the antistatic bag when they are not in use or not installed in the chassis.

Some circuitry on the system board can continue operating even though the power is switched off. Under no circumstances should the Lithium coin cell be used to power the real-time clock be allowed to be shorted. The coin cell can heat under these conditions and present a burn hazard.

WARNING!

1. "CAUTION: Danger of explosion if battery is incorrectly replaced. Replace only with the same or equivalent type recommended by the manufacturer. Discard used batteries according to the manufacturer's instructions"
2. This guide is for technically qualified personnel who have experience installing and configuring system boards. Disconnect the system board power supply from its power source before you connect/disconnect cables or install/remove any system board components. Failure to do this can result in personnel injury or equipment damage.
3. Avoid short-circuiting the lithium battery; this can cause it to superheat and cause burns if touched.
4. Do not operate the processor without a thermal solution. Damage to the processor can occur in seconds.
5. Do not block air vents. Minimum 1/2-inch for clearance required.

2.3 The Chassis

The system is integrated in a customized chassis (***Fig. 2-1, Fig. 2-2***). On the front panel you will find the Power LED, Hard Disk LED and LAN LED. The back panel has Five LAN ports and a COM port.

Fig. 2-1 *Front view of the Chassis*

Fig. 2-2 *Back view of the Chassis*

2.4 Open the Chassis

1. Take off the four screws (three at the rear side and two at the right/left side and remove the top lead (***Fig. 2-3***).

Fig. 2-3 *Take off two screws*

2. The top lead (***Fig. 2-4***) can be removed from the base stand (***Fig. 2-5***).

Fig. 2-4 *The top lead*

Fig. 2-5 *The base stand*

2.5 Install or Remove a SODIMM

Follow these steps to upgrade or remove RAM module:

1. Install the system memory by pulling the socket's arm and pressing it into the slot gently. (***Fig. 2-6, 2-7***)

Fig. 2-6 *The memory slot*

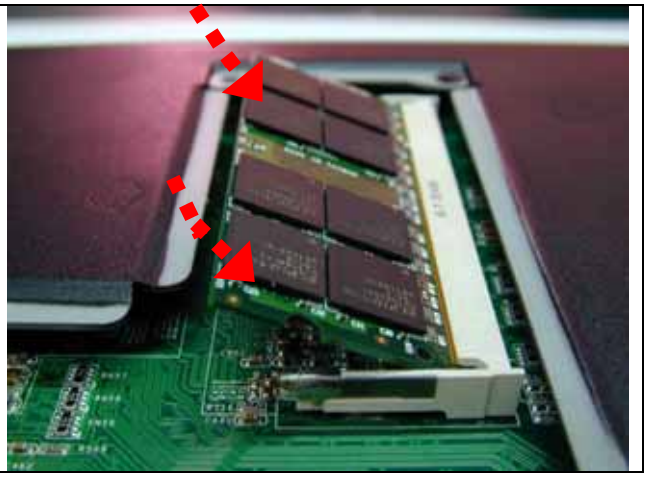


Fig. 2-7 *Install SODIMM*

2. By pulling the arms, the SODIMM can eject itself (***Fig. 2-8***).

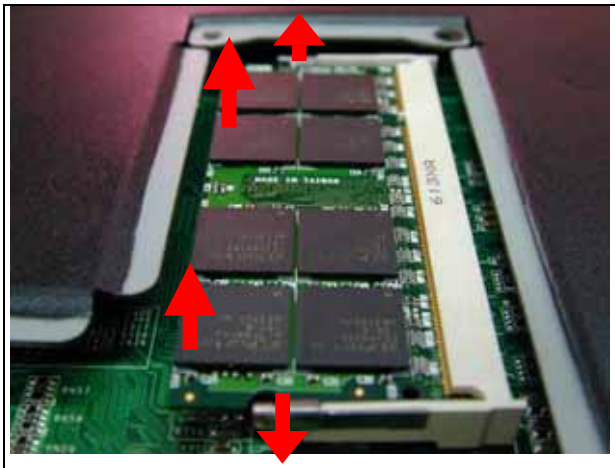


Fig. 2-8 *Eject a SODIMM module*

2.6 Remove and Install Battery

1. Press the metal clip back to eject the button battery (***Fig. 2-9***).
2. Replace it with a new one by pressing the battery with fingertip to restore the battery (***Fig. 2-10***).

Fig. 2-9 Eject *the battery*

Fig. 2-10 Restore *the battery*

2.7 Install Compact Flash

The system has an internal drive bay for one Compact Flash card drive. If the CF is not pre-installed, you can install it by yourself. Follow the steps below to install the CF:

3. Fasten the five screws to lock bracket together (***Fig. 2-11a, 2-11b***).

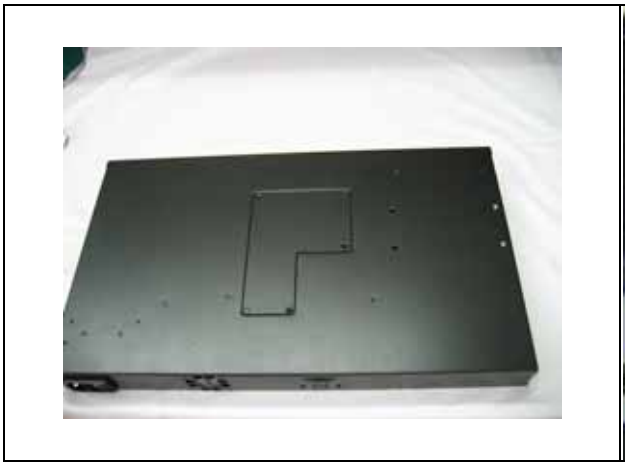
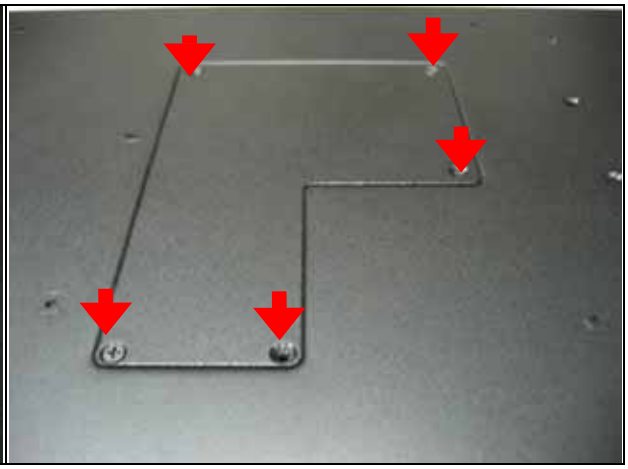


Fig. 2-11a *Remove L type base under button case*

Fig. 2-11b Push *CF into the bracket*

4. Completion CF to the System Chassis (***Fig. 2-12***)

Fig. 2-12 completion *CF in system*



Fix all screws back (***Fig. 2-13***).

2.8 Install 3.5" Hard disk

The system has an internal drive bay for one 3.5" hard disk drive. If the HDD is not pre-installed, you can install by yourself. You need the parts from the accessory-bag as shown on ***Figure 2-19***. They are one HDD-bracket, several screws. (from left to right).

(***Fig. 2-14***)	(***Fig. 2-15***) Fix the Metal Spacers (14mm)
(***Fig. 2-16***) Placement the HDD bracket	(***Fig. 2-17***). Fix HDD-bracket

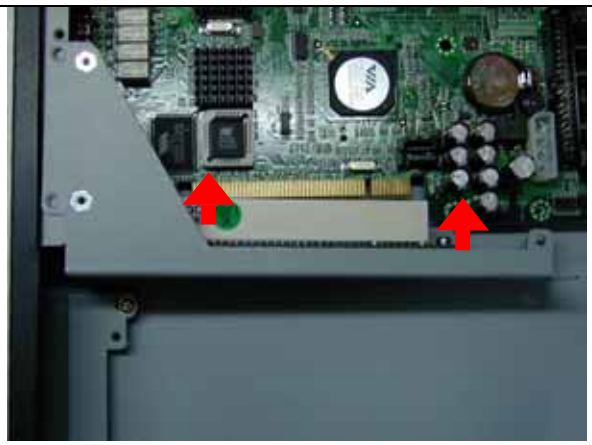
5. Connect Power Cable and IDE Cable before assemble hard disk. After assemble hard disk, put IDE Cable and Power cable into board.

(Fig. 2-18). Fix the hard disk drive on the HDD bracket with four screws. Plug the IDE cable into hard disk drive connector	***(Fig. 2-19***). Completion HDD with bracket.

2.9 Add riser card into system

1. Put Riser Card into Metal – bracket.
2. Put PCI Card Device into riser card assembly

(***Fig. 2-20***)



(***Fig. 2-21)*** Fix the Metal Spacers (14mm)

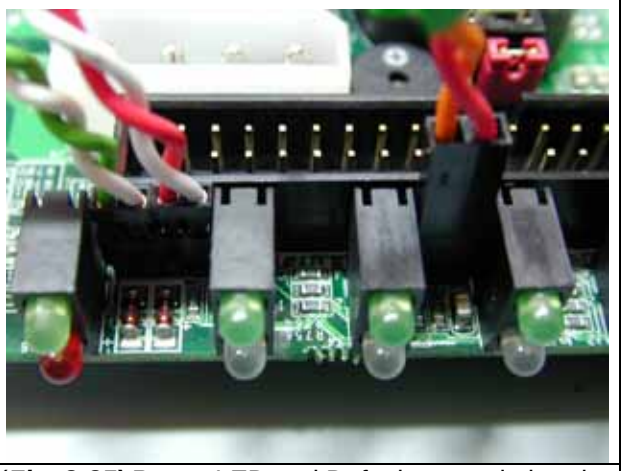
(***Fig. 2-22***) Fix the riser card bracket

(***Fig. 2-23***.)

2.10 Default Reset cable & Status cable

1. insert HDD status cable
2. Insert Power status cable
3. insert Default Reset cable

(***Fig. 2-24***)Pin header location



(***Fig. 2-25)*** Power,LED and Default reset pin header

2.11 Product Specifications

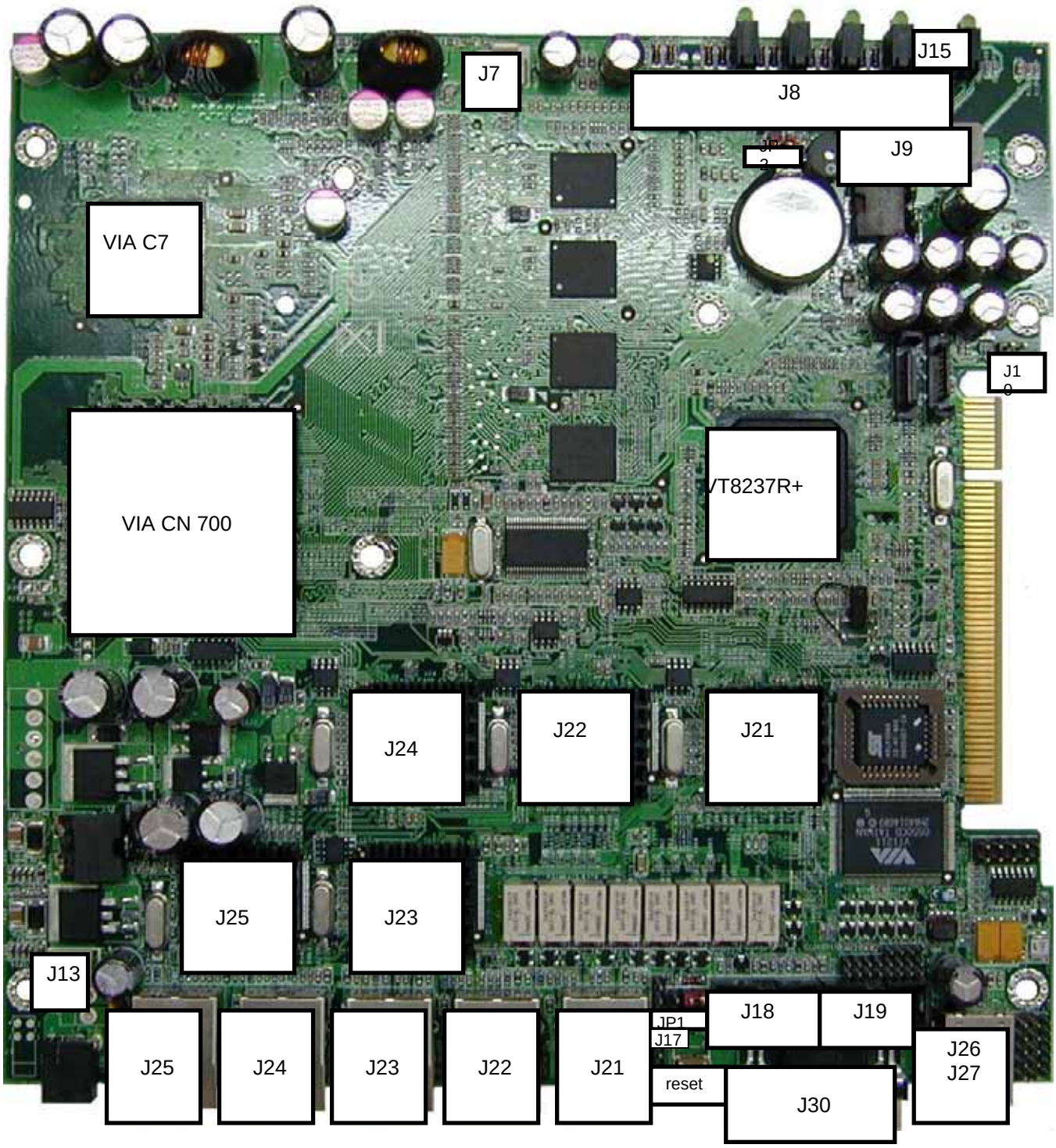
Model:	**NAR-2090**
Main Processor:	• VIA C7 processors (FSB 400/533Mhz)
BIOS:	• Award system BIOS with 512KB flash ROM to support DMI, PnP, APM function
Main Memory:	• Up to 1024MB 1.8V DDR2
L2 Cache Memory:	• 128KB 32-way built in (C7) CPU module
Chipset:	• VIA 8237R+
SATA Interface	• Two SATA I DMA133 Storage
PCI IDE Interface:	• One 40 Pin for DMA/33/66/100 IDE Storage
Serial Ports:	• Support two high-speed 16550 compatible UARTs with 16-byte T/R FIFOs
USB Interface:	• Support two USB2.0 ports for high speed I/O peripheral devices
Auxiliary I/O Interfaces:	• System reset switch, power okay LED, Ethernet activity LED, Ethernet speed LED, general purpose LED, alert LED and HDD LED interface
Power Input:	• Support one AC Adaptor with Adaptor input (power requirement: Input: 100-240V, Output: 15V == 4A)
On-board Ethernet:	• Five RealTek 8100C+ 10BASE-T/100BASE-TX Fast Ethernet controller with RJ-45 interface • Five RealTek 8110SC 32-bit Gb Ethernet controller with RJ-45 interface
Hardware Monitor:	Support on-board hardware monitor for • CPU fan, System fan • System voltages
Environmental Requirements:	• Operating Temperature: 5°C ~ 40°C • Storage Temperature: 0°C ~ 70°C • Relative Humidity: 5% ～ 95%, non-condensing
Dimension:	• 214mm(L) x 225mm (W) x 52mm (H)

2.12 Hardware Configuration Setting

This section gives the definitions and shows the positions of jumpers, headers and connectors. All of the configuration jumpers on PPAP-2011 are in the proper position. The default settings set by factory are marked with a star (★).

Jumpers

In general, jumpers on PPAP-2020 system board are used to select options for certain features. Some of the jumpers are configurable for system enhancement. The others are for testing purpose only and should not be altered. To select any option, cover the jumper cap over (Short) or remove (NC) it from the jumper pins according to the following instructions. Here NC stands for “Not Connected”.



PPAP-2020 Jumper Table

JP1 & J17 pin Header

2	4	6
1	3	5
+	-	

JP1	Function
1-3 , 2-4 Short	Power on default at Normal mode , mode selection is S/W programmable★

3-5 , 2-4 Short	Power on default at Bypass mode , mode selection is S/W programmable
4-6 Short	Always at normal mode.

JP2: CMOS Clear

JP2	Function
1-2 Short	Normal Operation ★
2-3 Short	Clear CMOS Contents

J10: Reset to default function

J10	Function
1-2 Short	RESET TO DEFAULT
1-2 Open	Normal mode ★

Connector	Function	Remark
J1~J5	LAN LED	
J7	CPU FAN connector	
J8	IDE connector	
J9	+5V & +12V power connector	
J10	RESET TO DEFAULT	
J13	SYS FAN connector	
J14	8-bit GPO LED connector	
J15	HDD LED +Power LED	
J17	By-pass LED	
J18	VGA connector	
J19	COM2 connector	
J21~J24	RJ45 connector	
J26、J27	USB connector	
J30	COM1 connector	
J31	PICMG 1.0 connector	

SW1: System reset

PIN No.	Signal Description
1	RST_SW
2	Ground

J7/J13: Fan power connector

PIN No.	Signal Description
1	GND
2	+12V
3	RPM signal

J18: On-board VGA 2x5 shrouded connector

PIN No.	Signal Description	PIN No.	Signal Description
1	RED	**2**	Green
3	Blue	**4**	VSYNC
5	HSYNC	**6**	SPCLK
7	Ground	**8**	Not connected
9	DDC data	**10**	HSYNC

J21/J22/J23/J24/J25: Ethernet5 RJ-45 interface connector

PIN No.	Signal Description
1	NC
2	LAN_MD0+
3	LAN_MD0-
4	TC1
5	NC
6	NC
7	TC2
8	LAN_MD1+
9	LAN_MD1-
10	NC
11	ACTIVE#
12	LINK#
13	LINK100#
14	+3.3V
15	Ground
16	Ground

J19: Serial port 2x5 shrouded connector (COM2)

PIN No.	Signal Description
1	DCD, Data carrier detect
2	DSR, Data set ready
3	RXD, Receive data
4	RTS, Request to send
5	TXD, Transmit data
6	CTS, Clear to send
7	DTR, Data terminal ready
8	RI, Ring indicator
9	GND, Ground
10	NC, Not connected

J8: IDE1 2x20 shrouded connector

PIN No.	Signal Description	PIN No.	Signal Description
1	Reset IDE	21	DRQ0
2	Ground	22	Ground
3	Host data 7	23	Host IOW
4	Host data 8	24	Ground
5	Host data 6	25	Host IOR
6	Host data 9	26	Ground
7	Host data 5	27	IOCHRDY0 {IOCHRDY1}
8	Host data 10	28	Not connected
9	Host data 4	29	DACK0 {DACK1}
10	Host data 11	30	Ground
11	Host data 3	31	IRQ14
12	Host data 12	32	Not connected
13	Host data 2	33	Address 1
14	Host data 13	34	Not connected
15	Host data 1	35	Address 0
16	Host data 14	36	Address 2
17	Host data 0	37	Chip select 0
18	Host data 15	38	Chip select 1
19	Ground	39	Activity
20	Not connected	40	Ground

J30: Serial port D-SUB9 connector (COM1)

PIN No.	Signal Description
1	Data Carrier Detect (DCD)
2	Receive Data (RXD)
3	Transmit Data (TXD)
4	Data Terminal Ready (DTR)
5	Ground (GND)
6	Data Set Ready (DSR)
7	Request to Send (RTS)
8	Clear to Send (CTS)
9	Ring Indicator (RI)

J26/J27: Dual USB port connector

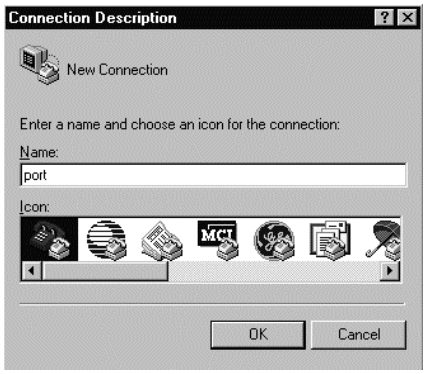
PIN No.	Signal Description	PIN No.	Signal Description
1	+5V	**2**	N/C
3	USBD0-	**4**	Ground
5	USBD0+	**6**	USBD1+
7	Ground	**8**	USBD1-
9	N/C	**10**	+5V

Connection Using Hyper Terminal

If users use a headless NAR-2090, which has no mouse/keyboard and VGA output connected to it, the console may be used to communicate with NAR-2090.

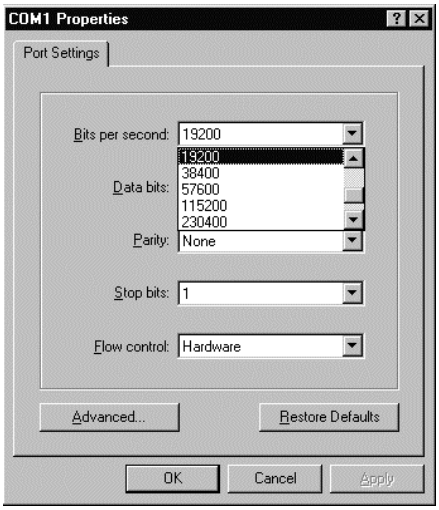
To access NAR-2090 via the console, Hyper Terminal is one of the choices. Follow the steps below for the setup:

1. Execute HyperTerminal under C:\Program Files\Accessories\HyperTerminal
2. Enter a name to create new dial

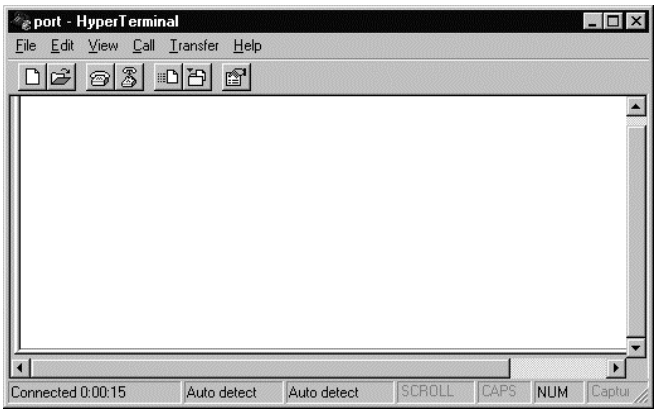


3. For the connection settings, make it Direct to COM1.

4. Please make the port settings to Baud rate 19200, Parity None, Data bits 8, Stop bits 1



5. Turn on the power of NAR-2090, after following screen was shown



6. You can then see the boot up information of NAR-2090

   When message “Hit <DEL> if you want to run Setup” appear during POST, after turning on or rebooting the computer, press **<Tab>** key ***immediately*** to enter BIOS setup program.

7. This is the end of this section. If the terminal did not port correctly, please check the previous steps.

2.14 BIOS Setup Information

NAR-2090 is equipped with the Award BIOS within Flash ROM. The BIOS has a built-in setup program that allows users to modify the basic system configuration easily. This type of information is stored in CMOS RAM so that it still retains during power-off periods. When system is turned on, NAR-2090 communicates with peripheral devices and checks its hardware resources against the configuration information stored in the CMOS memory. Whenever an error is detected, or the CMOS parameters need to be initially defined, the diagnostic program will prompt the user to enter the Setup program. Some errors are significant enough to abort the start-up.

Entering Setup

When you see the message “Hit <DEL> if you want to run Setup”, after turning on or rebooting the computer, press <Del> key ***immediately*** to enter BIOS setup program.

If you want to enter Setup but fail to respond before the message disappears, please restart the system either by first turning it off and followed by turning it on (COLD START) or simply press the "RESET" button. “WARM START” (press <Ctrl>, <Alt>, and <Delete> keys simultaneously) will do, too. Unless you press the keys at the right time, the system will not boot, an error message will display and you will be asked to do it again.

When no setting is stored in BIOS or the setting is missing, a message “Press <F1> to run Setup” will appear. Then press <F1> to run Setup or resume HIFLEX BIOS Setup. You can use the keyboard to choose among options or modify the system parameters to match the options with your system. The table shown on next page will show you all of keystroke functions in BIOS Setup.

Keys to navigate within Setup menu

Key	Function
Up (↑)	Move to the previous item
Down (↓)	Move to the next item
Left (→)	Move to the item on the left (menu bar)
Right (←)	Move to the item on the right (menu bar)
Enter	Enter the item you desired
PgUp	Increase the numeric value or make changes
PgDn	Decrease the numeric value or make changes
+	Increase the numeric value or make changes
—	Decrease the numeric value or make changes
Esc	**Main Menu:** Quit and not save changes into CMOS **Status Page Setup Menu and Option Page Setup Menu:** Exit current page and return to Main Menu
F1	General help on SETUP navigation keys
F5	Load previous values from CMOS
F6	Load the fail-safe defaults from BIOS default table
F7	Load the optimized defaults
F10	Save all the CMOS changes and exit

Main Menu

Once you enter NAR-2090 Award BIOS CMOS Setup utility, you should start with the Main Menu. The Main Menu allows you to select from eleven setup functions and two exit choices. Use arrow keys to switch among items and press <Enter> to accept or bring up the sub-menu.

Phoenix – Award BIOS CMOS Setup Utility

CMOS Setup Utility	
Standard CMOS Features Advanced BIOS Features Advanced Chipset Features Integrated Peripherals Power Management Setup PnP/PCI Configurations PC Health Status	Frequency /Voltage Control Load Fail-Safe Defaults Load Optimized Defaults Set Supervisor Password Set User Password Save & Exit Setup Exit Without Saving
ESC: Quit F10: Save & Exit Setup	↑ ↓ ← →: Select Item (Shift) F2: Change Color
Time, Date, Hard Disk Type ...	

NOTE: It is strongly recommended to reload the optimized default setting if CMOS is lost or BIOS is updated.

Standard CMOS Setup Menu

This setup page includes all the items within standard compatible BIOS. Use the arrow keys to highlight the item and then use the <PgUp>/<PgDn> or <+>/<-> keys to select the value or number you want in each item and press <Enter> to certify it.

Follow command keys in CMOS Setup table to change *Date, Time, Drive type* and *Boot Sector Virus Protection Status*.

Screen Shot: Phoenix – Award BIOS CMOS Setup Utility

Standard CMOS Setup Utility	
Date:(mm:dd:yy) Wed, Jan 05 2005 Time:(hh:mm:ss) 16:51:13 IDE Primary Master [None] IDE Primary Slave [None] IDE Secondary Master [None] IDE Secondary Slave [None] Video: EGA/VGA Base Memory: 640K Extended Memory: 95232K Total Memory: 96256K	
ESC: Quit F1: Help PU/PD/+/-: Modify	↑ ↓ ← →: Select Item (Shift) F2: Change Color

Menu Selections

Item	Options	Description
Date	mm:dd:yy	Set the system date. Note that the 'Day' automatically changes when you set the date
Time	hh:mm:ss	Set the system time
Video	EGA/VGA CGA 40CGA 80MONO	Select the default video device
Base Memory	N/A	Display the amount of conventional memory detected during boot up
Extended Memory	N/A	Display the amount of extended memory detected during boot-up
Total Memory	N/A	Display the total memory available in the system

Advance BIOS Features

This section allows user to configure your system for basic operation. Users will be able to select the system's default speed, boot-up sequence, keyboard operation, shadowing and security.

Screen Shot: Phoenix – Award BIOS CMOS Setup Utility

Advanced BIOS Features	
	ESC: Quit ↑ ↓ ← →: Select Item F1: Help (Shift) F2: Color F5: Old Values F6: Load BIOS Default F7: Load Setup Default PU/PD/+/-: Modify

Internal Cache/External Cache

These two categories speed up memory access. However, it depends on CPU/chipset design.

Enabled	Enable cache
Disabled	Disable cache

Quick Power On Self Test

This category speeds up Power On Self Test (POST) after you power up the computer. If it is set to Enable, BIOS will shorten or skip some check items during POST.

Enabled	Enable quick POST
Disabled	Normal POST

Boot Up NumLock Status

Select power on state for NumLock.

The choice: Enabled/Disabled.

Gate A20 Option

This entry allows user to select how the gate A20 is handled. The gate A20 is a device used to address memory over 1 Mbytes. Originally, the gate A20 was handled via a pin on the keyboard. But now, though keyboards still provide this support, it is more common, and much faster, for the system chipset to provide support for gate A20.

Normal	Keyboard
Fast	Chipset

Typematic Rate Setting

Keystrokes repeat at a rate determined by the keyboard controller. When enabled, the typematic rate and typematic delay can be selected.

The choice: Enabled/Disabled.

Typematic Rate (Chars/Sec)

Set the how many number of times a second to repeat a keystroke when a key is holding down.

The choice: 6, 8, 10, 12, 15, 20, 24 and 30.

Typematic Delay (Msec)

Set the delay time after the key is held down before it begins to repeat the keystroke.

The choice: 250, 500, 750 and 1000.

Security Option

Select whether the password is required every time the system boots or only when you enter setup.

System	The system will not boot and access to Setup will be denied if the correct password is not entered at the prompt.
Setup	The system will boot and access to Setup will be denied if the correct password is not entered at the prompt.

Note: To disable security, select PASSWORD SETTING at Main Menu and then user will be asked to enter password. Do not type anything and simply press <Enter>, it will disable security. Once the security is disabled, the system will boot up and user can enter Setup freely.

OS Select for DRAM > 64MB

Select the operating system that is running with more than 64MB of RAM on the system.

The choice: Non-OS2, OS2.

Console Redirection

Set the UNIX Console redirect to the terminal from COM1**.**

The choice: Enabled/Disabled.

Baud Rate

Set the RS-232 baud rate speed.

The choice: 9600, 19200, 38400, 57600 and 115200.

Advanced Chipset Features

This section allows user to configure your system for AT clock, DRAM timings...

Onboard LAN BootROM

This function decide whether invoke the boot ROM of the onboard Realtek LAN chip.

Disable: Disabled this function. (Default value)
Enable : Enabled this function. (Default value)

2.15 Reset to Default Sample Code information

```
//
//  Portwell Confidential !
//  Portwell Intellectual Property,  All  rights  reserved.
//
//  First release : 07/25/2006, 2020BP00.EXE, for ZR1 PCBA, By Frank Hsu
//
//  Second release: 09/08/2006, 2020BP01.EXE, for ZR2 and later PCBA, Modified by Frank Hsu
//  Add SYSWDT support.  SYSWDT has the similar programming algorithm with BP_WDT.
//
////////////////////////////////////////////////////////////////////////////
//
//     Program   : 2020BP01.CPP
//     Descript. : BY PASS test program for PPAP-2020
//     Designer  : Frank Hsu
//     Language  : Borland C++ 5.02
//     O.S.      : MS-DOS/Win98  only
//     Updddate  : 09082006  Release
//
//
////////////////////////////////////////////////////////////////////////////
//
//
//
//   Programming  guide :
//   Step1 : Define the GPIO pins for Bypass features
//         (1-1) SIOCFG_Rx24_bit7P1
//         (1-2) SIOCFG_Rx25_bit5P1
//   Step2 : Point to Logic Device(LD) number8 for GPIO and init these GPIO pins
//         LD8_RxF0 : Port Selection  , Bit[2..0]: [000] --> Port1 , (GP10~17)
//                                                 [001] --> Port3 , (GP30~37)
//                                                 [010] --> Port4 , (GP40~47)
//                                                 [011] --> Port5 , (GP50~57)
//                                                 [100] --> Port6 , (GP60~67)
//         LD8_RxF1 : Pin  direction , 1=output, 0=input.
//         LD9_RxF2 : Pin Polarity ,   0=Normal(not inverted), 1=inverted
//         (2-1) SIOCFG_Rx07P08
//         (2-2) LD8_Rx30_Bit0P1
//         (2-3) Get GPIOBASE
//           GPIOBASE is composed from LD8_Rx60 ( High byte ) and _Rx61 ( Low byte )
//           Address Offset 00 from GPIOBASE  is for  GPIO port1 pins.
```

```
//          Address Offset 01 from GPIOBASE  is for  GPIO port3 pins.
//          Address Offset 02 from GPIOBASE  is for  GPIO port4 pins.
//          Address Offset 03 from GPIOBASE  is for  GPIO port5 pins.
//
//        (2-4) LD8_RxF0_Bit[2..0]P[001]
//              LD8_RxF2P00
//              LD8_RxF1_Bit[7,5,4]P[111]
//        (2-5) LD8_RxF0_Bit[2..0]P[010]
//              LD8_RxF2P00
//              LD8_RxF1PF3
//        (2-6) LD8_RxF0_Bit[2..0]P[011]
//              LD8_RxF2P00
//              LD8_RxF1_Bit[1,0]P[11]
//        (2-7) LD8_RxF0_Bit[2..0]P[000]
//              LD8_RxF2P00
//              LD8_RxF1_Bit7P1
//              (GPIOBASE + 00)_Bit7P0
//   Must be enabled and write 0 to the GPIO pin : GPO17
//
//   Step3 : R/W GPIO data bits  from GPIO data port
//
//   BP_WDT[5..0]    : GPO51,50,47,46,45,44
//   LOADJ           : GPO34
//   BY_WDT_CLEARJ   : GPO35
//   BY_WDT_DISABLEJ : GPO41
//   BY_WDT_STATUS   : GPI42
//   BY_MODE_STATUS  : GPI43
//   SET_TO_NORMALJ  : GPO37
//   SET_TO_BYPASSJ  : GPO40
//
//   SYSWDT[5..0]    : GPO15,14,13,12,11,10
//   SYSWDT_CLEARJ   : GPO32
//   SYSWDT_DISABLEJ : GPO30
//   SYSWDT_STATUS   : GPI31
//
//
// << How  to  access SuperIO configuration Register >>
//
//   config_VT1211 : Index Port to access superIO config. register = 002eh
//
//
//   Enter VT1211  extended  mode to access CR ( Configuartion Register )
//      (config_VT1211)_P87 ;
//      (config_VT1211)_P87 ;
//
//   Exit  VT1211  extended  mode to access CR
//      (config_VT1211)_Paa ;
//
// <<<<<  How  to Set to Normal Mode  >>>>>
//
//             ( GPIOBASE + 01 )_Bit7P1
//             Delay 15us
//             ( GPIOBASE + 01 )_Bit7P0
//             Delay 15us
//             ( GPIOBASE + 01 )_Bit7P1
//             Delay 4 ms
//
//              --|_|---------
//               H-L-H pusle
```

```
//            For First High duration, around 15us is enough.
//            For  low  duration , 15us is enough.
//            For second High duration, it needs at least 4 ms to operate
//            the mechanical relay contacts.
//
// <<<<< How to Set to Bypass Mode >>>>>
//
//            ( GPIOBASE + 02 )_Bit0P1
//            Delay 15us
//            ( GPIOBASE + 02 )_Bit0P0
//            Delay 15us
//            ( GPIOBASE + 02 )_Bit0P1
//            Delay 4 ms
//
//             --|_|---------
//              H-L-H pusle
//            For First High duration, around 15us is enough.
//            For  low  duration , 15us is enough.
//            For second High duration, it needs at least 4 ms to release
//            the mechanical relay contacts.
//
//   <<<<< How to Read BP mode  >>>>>
//
//  Read BP status bit. "0" ---> Normal mode
//                      "1" ---> Bypass mode.
//     ( GPIOBASE + 02 )_Bit3
//
//
//    <<<<< How to Enable/Refresh BP_WDT >>>>>
//   Set Twd First: ( GPIOBASE + 03 )_Bit[1,0] --> BP_WDT[5,4].
//                  ( GPIOBASE + 02 )_Bit[7..4] --> BP_WDT[3..0]
//                          BP_WDT[5..0] : 000001b ~ 111111b
//                          ( "1 ~ 63 seconds" options available )
//
//                  Delay 15us
//   Make a H-L-H pulse for loadJ signal :
//                  ( GPIOBASE + 01 )_Bit4P1
//                  Delay 15us
//                  ( GPIOBASE + 01 )_Bit4P0
//                  Delay 15us
//                  ( GPIOBASE + 01 )_Bit4P1
//                  Delay 15us
//
//              --|_|--
//                H-L-H pusle
//              For  High duration, around 15us is enough.
//              For  low  duration , 15us is enough.
//
//   <<<<< How to Disable BP_WDT >>>>>
//
//      Make a H-L-H pulse with 15us width for disabling BP_WDT.
//       ( GPIOBASE + 02 )_Bit1
//
//    <<<<< How to Read BP_WDT status >>>>>
//    Read BP_WDT status bit. Normal ( WDT not expired ) , this bit returns "1".
//  If  BP_WDT expires, then this bit will be changed to "0".
//  ( Once BP_WDT expires , this bit will  keep "0".
//      When a power off/on
//         or writing to WDT_clear bit H-L-H pulse with 15us
```

```
//      then BP_WDT will be set to "1"  again. )
//
//                        BP_Group
//   -------------------+---------------------------------------
//    BP_WDT status bit   ( GPIOBASE + 02 )_Bit2
//   -------------------------------------------------------
//    BP_WDT clear bit    ( GPIOBASE + 01 )_Bit5   H-L-H pulse
// -------------------------------------------------------
//
//  ************************************************************************
//  SYSWDT programming guide is similar with BP_WDT.
//  ( Enable/Refresh SYSWDT, Disable SYSWDT, and read SYSWDT_SYS. )
//  If  SYSWDT  expires, then  PPAP-2020  will be  reset  immediately.
//  SYSWDT is implemented on ZR2 and later PCBA.
//

#include "stdlib.h"
#include "conio.h"
#include "stdio.h"
#include "dos.h"                  // for delay(), and sleep()

#pragma  inline                   // for inline asm

// Global  constant --------- Start --------------
#define GPIO1X_OFFSET    0x00   // The offset value from GPIOBASE
#define GPIO3X_OFFSET    0x01   // The offset value from GPIOBASE
#define GPIO4X_OFFSET    0x02   // The offset value from GPIOBASE
#define GPIO5X_OFFSET    0x03   // The offset value from GPIOBASE
#define GPIO6X_OFFSET    0x04   // The offset value from GPIOBASE

#define config_VT1211    0x2E   // Hardware strapping
#define GPIO_LDN         0x08   // GPIO LDN = 0x08
#define portb            0x61
#define refresh_status   0x10
#define CRF0             0xF0
#define CRF1             0xF1
#define CRF2             0xF2
#define Non_inversed_byte 0x00  // 8 GPIO pins  not  inversed.
// Global  constant --------- End  --------------


// Global Variable ----- Start ---
     unsigned  int  GPIOBASE  ;
     struct  time t;
     struct  date d;

   //  char *VER_DATE = "2020BP00.exe, V1.00 07-25-2006" ; // First release
     char *VER_DATE = "2020BP01.exe, V1.00 09-08-2006" ; // Second release

// Global  Variable ---- end -----


void  IO_delay()
 {
   inportb(0x80) ;
   inportb(0x80) ;
 }

  void fixdelay_15us ()
```

```
{
  //  delay  15 us
  unsigned  char  char_ah,char_al ;
    char_ah = inportb ( portb ) & refresh_status ;
  fixdelay_loop :
    char_al = inportb ( portb ) & refresh_status ;
    if(char_ah == char_al ) goto fixdelay_loop ;
} // end of fixdelay_15us



void WDT_disable()
{
 //  Global Variable GPIOBASE needs to be known first.
 //  Global  constant  adopted.
 //  Called function : IO_delay() , fixdelay_15us() .
  unsigned char  al_3 ;
  al_3 = inportb( GPIOBASE + GPIO4X_OFFSET ) | 0x02   ; // GP41 : WDT_disable
  IO_delay() ;
  outportb( GPIOBASE + GPIO4X_OFFSET , al_3 ) ; // P1  , First high pulse
  fixdelay_15us() ;
  al_3 = inportb( GPIOBASE + GPIO4X_OFFSET ) & 0xFD  ;
  IO_delay() ;
  outportb( GPIOBASE + GPIO4X_OFFSET , al_3 ) ; // P0 , low pulse
  fixdelay_15us() ;
  al_3 = inportb( GPIOBASE + GPIO4X_OFFSET ) | 0x02   ;
  IO_delay() ;
  outportb( GPIOBASE + GPIO4X_OFFSET , al_3 ) ; // P1  , Second high pulse
  fixdelay_15us() ;

}  // end of WDT_disable()

// ------------------------------------
void SYSWDT_disable()
{
 //  Global Variable GPIOBASE needs to be known first.
 //  Global  constant  adopted.
 //  Called function : IO_delay() , fixdelay_15us() .
  unsigned char  al_3 ;
  al_3 = inportb( GPIOBASE + GPIO3X_OFFSET ) | 0x01   ; // GP30 : WDT_disable
  IO_delay() ;
  outportb( GPIOBASE + GPIO3X_OFFSET , al_3 ) ; // P1  , First high pulse
  fixdelay_15us() ;
  al_3 = inportb( GPIOBASE + GPIO3X_OFFSET ) & 0xFE  ;
  IO_delay() ;
  outportb( GPIOBASE + GPIO3X_OFFSET , al_3 ) ; // P0 , low pulse
  fixdelay_15us() ;
  al_3 = inportb( GPIOBASE + GPIO3X_OFFSET ) | 0x01   ;
  IO_delay() ;
  outportb( GPIOBASE + GPIO3X_OFFSET , al_3 ) ; // P1  , Second high pulse
  fixdelay_15us() ;

}  // end of SYSWDT_disable()


 void LOADJ_HLH()
 {
 //  Global Variable GPIOBASE needs to be known first.
 //  Global  constant  adopted.
```

```
 //  Called function : IO_delay() , fixdelay_15us() .
  unsigned char al_char ;

// LOADJ  H_L_H  pulse ---------------------------------------------
  al_char = inportb( GPIOBASE + GPIO3X_OFFSET ) | 0x10   ; // GP34 : LOADJ
  IO_delay() ;
  outportb( GPIOBASE + GPIO3X_OFFSET , al_char ) ; // P1  , First high pulse
  fixdelay_15us() ;
  al_char = inportb( GPIOBASE + GPIO3X_OFFSET ) & 0xEF  ;
  IO_delay() ;
  outportb( GPIOBASE + GPIO3X_OFFSET , al_char ) ; // P0 , low pulse
  fixdelay_15us() ;
  al_char = inportb( GPIOBASE + GPIO3X_OFFSET ) | 0x10   ;
  IO_delay() ;
  outportb( GPIOBASE + GPIO3X_OFFSET , al_char ) ; // P1  , Second high pulse
  fixdelay_15us() ;
  // ------------------------------------------------
} // End of LOADJ_HLH()


 void SYSWDT_LOADJ_HLH()
{
 //  Global Variable GPIOBASE needs to be known first.
 //  Global  constant  adopted.
 //  Called function : IO_delay() , fixdelay_15us() .
  unsigned char al_char ;

// LOADJ  H_L_H  pulse ---------------------------------------------
  al_char = inportb( GPIOBASE + GPIO1X_OFFSET ) | 0x40   ; // GP16 : LOADJ
  IO_delay() ;
  outportb( GPIOBASE + GPIO1X_OFFSET , al_char ) ; // P1  , First high pulse
  fixdelay_15us() ;
  al_char = inportb( GPIOBASE + GPIO1X_OFFSET ) & 0xBF  ;
  IO_delay() ;
  outportb( GPIOBASE + GPIO1X_OFFSET , al_char ) ; // P0 , low pulse
  fixdelay_15us() ;
  al_char = inportb( GPIOBASE + GPIO1X_OFFSET ) | 0x40   ;
  IO_delay() ;
  outportb( GPIOBASE + GPIO1X_OFFSET , al_char ) ; // P1  , Second high pulse
  fixdelay_15us() ;
  // ------------------------------------------------
} // End of SYSWDT_LOADJ_HLH()

 void CLEARJ_BP_WDT_STS_HLH()
{
 //  Global Variable GPIOBASE needs to be known first.
 //  Global  constant  adopted.
 //  Called function : IO_delay() , fixdelay_15us() .
  unsigned char al_char ;
  // CLEARJ  H_L_H  pulse ---------------------------------------------
  al_char = inportb( GPIOBASE + GPIO3X_OFFSET ) | 0x20   ; // GP35 : CLEARJ
  IO_delay() ;
  outportb( GPIOBASE + GPIO3X_OFFSET , al_char ) ; // P1  , First high pulse
  fixdelay_15us() ;
  al_char = inportb( GPIOBASE + GPIO3X_OFFSET ) & 0xDF  ;
  IO_delay() ;
  outportb( GPIOBASE + GPIO3X_OFFSET , al_char ) ; // P0 , low pulse
  fixdelay_15us() ;
  al_char = inportb( GPIOBASE + GPIO3X_OFFSET ) | 0x20   ;
```

```
  IO_delay() ;
  outportb( GPIOBASE + GPIO3X_OFFSET , al_char ) ; // P1  , Second high pulse
  fixdelay_15us() ;
  // ------------------------------------------------
 } // CLEARJ_BP_WDT_STS_HLH()

 void CLEARJ_SYS_WDT_STS_HLH()
 {
  //  Global Variable GPIOBASE needs to be known first.
  //  Global  constant  adopted.
  //  Called function : IO_delay() , fixdelay_15us() .
   unsigned char al_char ;
   // CLEARJ  H_L_H  pulse ----------------------------------------------
   al_char = inportb( GPIOBASE + GPIO3X_OFFSET ) | 0x04   ; // GP32 : CLEARJ
   IO_delay() ;
   outportb( GPIOBASE + GPIO3X_OFFSET , al_char ) ; // P1  , First high pulse
   fixdelay_15us() ;
   al_char = inportb( GPIOBASE + GPIO3X_OFFSET ) & 0xFB  ;
   IO_delay() ;
   outportb( GPIOBASE + GPIO3X_OFFSET , al_char ) ; // P0 , low pulse
   fixdelay_15us() ;
   al_char = inportb( GPIOBASE + GPIO3X_OFFSET ) | 0x04   ;
   IO_delay() ;
   outportb( GPIOBASE + GPIO3X_OFFSET , al_char ) ; // P1  , Second high pulse
   fixdelay_15us() ;
   // ------------------------------------------------
 } // CLEARJ_SYS_WDT_STS_HLH()

unsigned char WDT_REFRESH_1K( unsigned int ax_twd ,unsigned int ax_idle_ms )
{
  //  Global Variable GPIOBASE needs to be known first.
  //  Global  constant  adopted.
  //  Called function : IO_delay() ,WDT_disable(), fixdelay_15us(), LOADJ_HLH() , inline asm.
  //  Retuned : ESC return : 0x1b ; Normal ( 1000 times refresh ) return : 0x00.
  unsigned  char  al_4 , al_5 ;
  unsigned  int  i,j ;
  al_4 = ax_twd  ;  // Truncate high word.
  // Twd load into WDT[5:0] ( GP51,50,47,46,45,44 ) first ==============
  al_5 = inportb( GPIOBASE + GPIO5X_OFFSET ) & 0xFC ;
  al_5 = al_5 | ( al_4 >> 4 ) ;
  IO_delay() ;
  outportb( GPIOBASE + GPIO5X_OFFSET , al_5 ) ; // write GP51,50

  al_5 = inportb( GPIOBASE + GPIO4X_OFFSET ) & 0x0F ;
  al_5 = al_5 | ( al_4 << 4 ) ;
  IO_delay() ;
  outportb( GPIOBASE + GPIO4X_OFFSET , al_5 ) ; // write GP47,46,45,44

  for ( i = 1 ; i <=1000 ; i ++ )
  {
   LOADJ_HLH() ;

   for( j=1; j<=80 ; j++ ) printf("\b") ;

   printf("  Enable or Refresh BP_WDT %d times at", i );
   // Get  time and Date
   gettime(&t);
   printf(" %02d:%02d:%02d.%02d , ",
         t.ti_hour, t.ti_min, t.ti_sec, t.ti_hund);
```

```
  getdate(&d);
     printf("%d/%d/%d. Refreshing...",d.da_mon, d.da_day, d.da_year);
  // for( j=1; j<=80 ; j++ ) printf("\b") ;

  if( i == 1 ) delay (6) ;
  else   fixdelay_15us() ;
  if ( ax_idle_ms == 1 ) delay(ax_idle_ms);
  else if ( i ==1 ) delay( ax_idle_ms - 6) ;
  else delay(ax_idle_ms);

  al_5 = inportb( GPIOBASE + GPIO4X_OFFSET ) & 0x08 ;// Check Bypass status , GP43
  if ( al_5 == 0x00 ) goto okay_loop ;                // low (0) = normal
  else
  {
    printf("\n\n Error#01: Refresh BP_WDT 1000 times --> Failed, Check CPLD or H/W Circuit!!!\n");
    outportb ( config_VT1211 , 0xaa ) ;   // exit  config  mode
    exit(1) ;
  }
  okay_loop:
   asm  mov ah,6    ;      // check "Esc"  key  pressed ?
   asm  mov dl,0ffh ;
   asm  int 21h     ;      // Call DOS function call
   asm  cmp al,1bh  ;
   asm  je esc_return   ;  // if "Esc" pressed , goto  esc_return

  } // for i loop

  WDT_disable() ;


  return 0x00 ;

 esc_return :
  WDT_disable() ;
  return 0x1b ;


}// end of uc WDT_REFRESH_1K(ui,ui)

// ********************************************************************************

unsigned char WDT_RFSH_1K( unsigned int ax_twd ,unsigned int ax_idle_ms )
{
  //  Global Variable GPIOBASE needs to be known first.
  //  Global  constant  adopted.
  //  Called function : IO_delay() ,SYSWDT_disable(), fixdelay_15us(), SYSWDT_LOADJ_HLH() , inline asm.
  //  Retuned : ESC return : 0x1b ; Normal ( 1000 times refresh ) return : 0x00.
  unsigned  char  al_4 , al_5 ;
  unsigned  int  i,j ;
  al_4 = ax_twd & 0x3F  ;  // Truncate high word.
  // Twd load into WDT[5:0] ( GP15~10 ) first ==============
  al_5 = inportb( GPIOBASE + GPIO1X_OFFSET ) & 0xC0 ;
  al_5 = al_5 | al_4  ;
  IO_delay() ;
  outportb( GPIOBASE + GPIO1X_OFFSET , al_5 ) ; // write GP15~10


  for ( i = 1 ; i <=1000 ; i ++ )
  {
```

```
  SYSWDT_LOADJ_HLH() ;

  for( j=1; j<=80 ; j++ ) printf("\b") ;

  printf("  Enable or Refresh SYS_WDT %d times at", i );
  // Get  time and Date
  gettime(&t);
  printf(" %02d:%02d:%02d.%02d , ",
        t.ti_hour, t.ti_min, t.ti_sec, t.ti_hund);
  getdate(&d);
     printf("%d/%d/%d. Refreshing...",d.da_mon, d.da_day, d.da_year);
 // for( j=1; j<=80 ; j++ ) printf("\b") ;

   if( i == 1 ) delay (6) ;
  else   fixdelay_15us() ;
  if ( ax_idle_ms == 1 ) delay(ax_idle_ms);
  else if ( i ==1 ) delay( ax_idle_ms - 6) ;
  else delay(ax_idle_ms);

  al_5 = inportb( GPIOBASE + GPIO3X_OFFSET ) & 0x01 ;  // Check Bypass status , GP31
  if ( al_5 == 0x01 ) goto SYSWDT_okay_loop ;           // High(1) = normal
  else
  {
    printf("\n\n Error#0B: Refresh WDT 1000 times --> Failed, Check CPLD or H/W Circuit!!!\n");
    outportb ( config_VT1211 , 0xaa ) ;   // exit  config  mode
    exit(1) ;
  }
  SYSWDT_okay_loop:
   asm  mov ah,6    ;      // check "Esc"  key  pressed ?
   asm  mov dl,0ffh ;
   asm  int 21h     ;      // Call DOS function call
   asm  cmp al,1bh  ;
   asm  je SYSWDT_esc_return   ;  // if "Esc" pressed , goto  esc_return

  } // for i loop

  SYSWDT_disable() ;


  return 0x00 ;

 SYSWDT_esc_return :
  SYSWDT_disable() ;
  return 0x1b ;


}// end of uc WDT_RFSH_1K(ui,ui)


//
unsigned char Set_normal()
{
   //  Global Variable GPIOBASE needs to be known first.
   //  Global  constant  adopted.
   //  Called function : IO_delay() , fixdelay_15us() .
   //  test  okay : return 0x00
   //  Test failed : return 0xBB
   unsigned char  al_1 ;
   al_1 = inportb( GPIOBASE + GPIO3X_OFFSET ) | 0x80   ; // GP37 : Set_normal
```

```
  IO_delay() ;
  outportb( GPIOBASE + GPIO3X_OFFSET , al_1 ) ; // P1  , First high pulse

  fixdelay_15us() ;
  al_1 = inportb( GPIOBASE + GPIO3X_OFFSET ) & 0x7F  ;

  IO_delay() ;
  outportb( GPIOBASE + GPIO3X_OFFSET , al_1 ) ; // P0 , low pulse

  fixdelay_15us() ;
   al_1 = inportb( GPIOBASE + GPIO3X_OFFSET ) | 0x80   ;

  IO_delay() ;
  outportb( GPIOBASE + GPIO3X_OFFSET , al_1 ) ; // P1  , Second high pulse

  delay(10) ;  // delay 10 ms

  al_1 = inportb( GPIOBASE + GPIO4X_OFFSET ) & 0x08 ;// Check Bypass status , GP43
  if ( al_1 == 0x00 ) return (0x00) ;              // low (0) = normal
  else  return (0xBB) ;
} // end of  UC Set_normal


unsigned char Set_bypass()
{
   //  Global Variable GPIOBASE needs to be known first.
   //  Global  constant  adopted.
   //  Called function : IO_delay() , fixdelay_15us() .
   //  test  okay : return 0x00
   //  Test failed : return 0xBB
  unsigned char  al_2 ;
  al_2 = inportb( GPIOBASE + GPIO4X_OFFSET ) | 0x01   ; // GP40 : Set_bypass
  IO_delay() ;
  outportb( GPIOBASE + GPIO4X_OFFSET , al_2 ) ; // P1  , First high pulse
  fixdelay_15us() ;
  al_2 = inportb( GPIOBASE + GPIO4X_OFFSET ) & 0xFE  ;
  IO_delay() ;
  outportb( GPIOBASE + GPIO4X_OFFSET , al_2 ) ; // P0 , low pulse
  fixdelay_15us() ;
  al_2 = inportb( GPIOBASE + GPIO4X_OFFSET ) | 0x01   ;
  IO_delay() ;
  outportb( GPIOBASE + GPIO4X_OFFSET , al_2 ) ; // P1  , Second high pulse
  delay(10) ;  // delay 10 ms

  al_2 = inportb( GPIOBASE + GPIO4X_OFFSET ) & 0x08 ;// Check Bypass status , GP43
  if ( al_2 == 0x08 ) return (0x00) ;              // high(1) = bypass
  else  return (0xBB) ;


} // end of UC Set_bypass()

unsigned char Sub_menu_display ( unsigned char al_WDT_type , char al_sel )
{
     //  Global Variable , struct t and d adopted
     // al_WDT_type = 0x55 ---> BP_WDT
     //             = 0xAA ---> SYS_WDT
   unsigned int i ;
   for ( i=0 ; i < 26 ; i++ ) printf("\n") ;
```

```
  //   system ( "cls" );  Need DOS ( system, command.com ) exist

     if ( al_WDT_type == 0x55 )
     printf("\n Refresh BP_WDT 1000 times.    seconds : seconds ===> Twd : Refresh interval\n");
     else printf("\n Refresh SYS_WDT 1000 times.    seconds : seconds ===> Twd : Refresh interval\n");
     printf("\n < 1 >:  1 second :  0.989  second .   < A >: 38 seconds : 36.538 seconds.");
     printf("\n < 2 >:  2 seconds:  1.978  seconds.   < B >: 42 seconds : 40.385 seconds.");
     printf("\n < 3 >:  4 seconds:  3.956  seconds.   < C >: 46 seconds : 44.231 seconds.");
     printf("\n < 4 >:  8 seconds:  7.912  seconds.   < D >: 50 seconds : 48.077 seconds.");
     printf("\n < 5 >: 14 seconds: 13.461  seconds.   < E >: 54 seconds : 51.923 seconds.");
     printf("\n < 6 >: 20 seconds: 19.231  seconds.   < F >: 56 seconds : 53.846 seconds.");
     printf("\n < 7 >: 26 seconds: 25.000  seconds.   < G >: 58 seconds : 55.769 seconds.");
     printf("\n < 8 >: 30 seconds: 28.846  seconds.   < H >: 60 seconds : 57.692 seconds.");
     printf("\n < 9 >: 34 seconds: 32.692  seconds.   < I >: 62 seconds : 59.615 seconds.");
     printf("\n\n < Q >: RETURN  TO  MAIN  MENU.              Ts=");
     // Get  time and Date
     gettime(&t);
     printf(" %2d:%02d:%02d.%02d , ",
        t.ti_hour, t.ti_min, t.ti_sec, t.ti_hund);
     getdate(&d);
     printf("%d/%d/%d\n",d.da_mon, d.da_day, d.da_year);

     printf("\n   <<<<<<* OTHER  CHOICE  WILL  NOT  DELAY  *>>>>>>.\n");
     printf("\n   PRESS \"Esc\" KEY TO STOP REFRESHING BP_WDT & DISABLE BP_WDT.\n");
     if  ( al_sel != 'K' )   goto  all_display_1 ;
     if ( al_WDT_type == 0x55 )
     printf("\n\n   Your choice for BY_WDT refresh test : <_>\b\b") ;
     else   printf("\n\n   Your choice for SYS_WDT refresh test : <_>\b\b") ;
     return 0x00 ;
   all_display_1 :
     if ( al_WDT_type == 0x55 )
     printf("\n\n   Auto testing BP_WDT ....     Current choice : <%c>\n\n", al_sel ) ;
     else printf("\n\n   Auto testing SYS_WDT ....     Current choice : <%c>\n\n", al_sel ) ;
     return 0x01 ;
} // end of Sub_menu_display (c al_sel )


// ----
  void  P2020_GPIO_TEST()
 {
    unsigned char al_char , al_char1 ;
/*
     Testing  way :
  Use  PPAP-2020 ZR2 GPIO ( 8 bi-direction pins from Super IO VT1211 )
  Initialization for  VT1211  must be done first in Main().
  --- t1
  SGPO52  Write 0 to SGPI53 ,  SGPI53 = 0 ?  ,if yes, pass ; if no, failed
  SGPO54  Write 0 to SGPI55 ,  SGPI55 = 0 ?  ,if yes, pass ; if no, failed
  SGPO56  Write 0 to SGPI57 ,  SGPI57 = 0 ?  ,if yes, pass ; if no, failed
  SGPO60  Write 0 to SGPI61 ,  SGPI61 = 0 ?  ,if yes, pass ; if no, failed
  --- t2
  SGPO52  Write 1 to SGPI53 ,  SGPI53 = 1 ?  ,if yes, pass ; if no, failed
  SGPO54  Write 1 to SGPI55 ,  SGPI55 = 1 ?  ,if yes, pass ; if no, failed
  SGPO56  Write 1 to SGPI57 ,  SGPI57 = 1 ?  ,if yes, pass ; if no, failed
  SGPO60  Write 1 to SGPI61 ,  SGPI61 = 1 ?  ,if yes, pass ; if no, failed
  --- t3
  SGPO53  Write 0 to SGPI52 ,  SGPI52 = 0 ?  ,if yes, pass ; if no, failed
  SGPO55  Write 0 to SGPI54 ,  SGPI54 = 0 ?  ,if yes, pass ; if no, failed
```

```
 SGPO57  Write 0 to SGPI56 ,  SGPI56 = 0 ?  ,if yes, pass ; if no, failed
 SGPO61  Write 0 to SGPI60 ,  SGPI60 = 0 ?  ,if yes, pass ; if no, failed
 --- t4
 SGPO53  Write 1 to SGPI52 ,  SGPI52 = 1 ?  ,if yes, pass ; if no, failed
 SGPO55  Write 1 to SGPI54 ,  SGPI54 = 1 ?  ,if yes, pass ; if no, failed
 SGPO57  Write 1 to SGPI56 ,  SGPI56 = 1 ?  ,if yes, pass ; if no, failed
 SGPO61  Write 1 to SGPI60 ,  SGPI60 = 1 ?  ,if yes, pass ; if no, failed
*/
  // GPIO Direction  setting  for  t1 and t2  test  items ==============
   outportb ( config_VT1211 , CRF0 ) ;
   al_char = ( inportb ( config_VT1211+1 ) & 0xFB ) | 0x03 ; // Bit[2,1,0]P[011]
   outportb ( config_VT1211 , CRF0 ) ;                    // point to GPIO50~57
   outportb ( config_VT1211+1 , al_char ) ;

   outportb ( config_VT1211 , CRF1 ) ;
   al_char = ( inportb ( config_VT1211+1 ) & 0x57 ) | 0x54 ; // SGPO52,54,56 ;
   outportb ( config_VT1211 , CRF1 ) ;                    // SGPI53,55,57
   outportb ( config_VT1211+1 , al_char ) ;

   outportb ( config_VT1211 , CRF2 ) ;                    // Non Inversed GPIO (CRF2P00)
   outportb ( config_VT1211+1 , Non_inversed_byte ) ;



   outportb ( config_VT1211 , CRF0 ) ;
   al_char = ( inportb ( config_VT1211+1 ) & 0xFC ) | 0x04 ; // Bit[2,1,0]P[100]
   outportb ( config_VT1211 , CRF0 ) ;                    // point to GPIO60~67
   outportb ( config_VT1211+1 , al_char ) ;

   outportb ( config_VT1211 , CRF1 ) ;
   al_char = ( inportb ( config_VT1211+1 ) & 0xFD ) | 0x01 ; // SGPO60;
   outportb ( config_VT1211 , CRF1 ) ;                    // SGPI61
   outportb ( config_VT1211+1 , al_char ) ;

   outportb ( config_VT1211 , CRF2 ) ;                    // Non Inversed GPIO (CRF2P00)
   outportb ( config_VT1211+1 , Non_inversed_byte ) ;

  // t1 -----
   al_char = inportb( GPIOBASE + GPIO5X_OFFSET ) & 0xAB ;  // SGPO52,54,56 P[000]
   IO_delay() ;
   outportb( GPIOBASE + GPIO5X_OFFSET , al_char ) ;

   al_char = inportb( GPIOBASE + GPIO6X_OFFSET ) & 0xFE ;  // SGPO60P[0]
   IO_delay() ;
   outportb( GPIOBASE + GPIO6X_OFFSET , al_char ) ;

   fixdelay_15us() ;

   al_char = inportb( GPIOBASE + GPIO5X_OFFSET ) & 0xA8 ;  // Mask SGPI53,55,57
   al_char1= inportb( GPIOBASE + GPIO6X_OFFSET ) & 0x02 ;  // Mask SGPI61
   al_char = al_char | al_char1 ;

   if( al_char == 0x00 ) goto gp_ok_1 ;
   al_char1 = 0x01 ;
   goto  gpio_test_failed ;

  // t2 ------
  gp_ok_1 :
   al_char = inportb( GPIOBASE + GPIO5X_OFFSET ) | 0x54 ;  // SGPO52,54,56 P[111]
```

```
 IO_delay() ;
 outportb( GPIOBASE + GPIO5X_OFFSET , al_char ) ;

 al_char = inportb( GPIOBASE + GPIO6X_OFFSET ) | 0x01 ;  // SGPO60P[1]
 IO_delay() ;
 outportb( GPIOBASE + GPIO6X_OFFSET , al_char ) ;

 fixdelay_15us() ;

 al_char = inportb( GPIOBASE + GPIO5X_OFFSET ) & 0xA8 ;  // Mask SGPI53,55,57
 al_char1= inportb( GPIOBASE + GPIO6X_OFFSET ) & 0x02 ;  // Mask SGPI61
 al_char = al_char | al_char1 ;

 if( al_char == 0xAA )  goto  gp_ok_2 ;
 al_char1 = 0x02 ;
 goto gpio_test_failed ;

// GPIO Direction  setting  for  t3 and t4  test  items ==============
gp_ok_2 :
 outportb ( config_VT1211 , CRF0 ) ;
 al_char = ( inportb ( config_VT1211+1 ) & 0xFB ) | 0x03 ; // Bit[2,1,0]P[011]
 outportb ( config_VT1211 , CRF0 ) ;                    // point to GPIO50~57
 outportb ( config_VT1211+1 , al_char ) ;

 outportb ( config_VT1211 , CRF1 ) ;
 al_char = ( inportb ( config_VT1211+1 ) & 0xAB ) | 0xA8 ; // SGPI52,54,56 ;
 outportb ( config_VT1211 , CRF1 ) ;                    // SGPO53,55,57
 outportb ( config_VT1211+1 , al_char ) ;

 outportb ( config_VT1211 , CRF0 ) ;
 al_char = ( inportb ( config_VT1211+1 ) & 0xFC ) | 0x04 ; // Bit[2,1,0]P[100]
 outportb ( config_VT1211 , CRF0 ) ;                    // point to GPIO60~67
 outportb ( config_VT1211+1 , al_char ) ;

 outportb ( config_VT1211 , CRF1 ) ;
 al_char = ( inportb ( config_VT1211+1 ) & 0xFE ) | 0x02 ; // SGPI60;
 outportb ( config_VT1211 , CRF1 ) ;                    // SGPO61
 outportb ( config_VT1211+1 , al_char ) ;



// t3 ------------------
 al_char = inportb( GPIOBASE + GPIO5X_OFFSET ) & 0x57 ;  // SGPO53,55,57 P[000]
 IO_delay() ;
 outportb( GPIOBASE + GPIO5X_OFFSET , al_char ) ;

 al_char = inportb( GPIOBASE + GPIO6X_OFFSET ) & 0xFD ;  // SGPO61P[0]
 IO_delay() ;
 outportb( GPIOBASE + GPIO6X_OFFSET , al_char ) ;

 fixdelay_15us() ;

 al_char = inportb( GPIOBASE + GPIO5X_OFFSET ) & 0x54 ;  // Mask SGPI52,54,56
 al_char1= inportb( GPIOBASE + GPIO6X_OFFSET ) & 0x01 ;  // Mask SGPI60
 al_char = al_char | al_char1 ;

 if ( al_char == 0x00 ) goto gp_ok_3 ;
 al_char1 = 0x03 ;
 goto gpio_test_failed ;
```

```
  // t4 ------------------
   gp_ok_3 :
   al_char = inportb( GPIOBASE + GPIO5X_OFFSET ) | 0xA8 ;  // SGPO53,55,57 P[111]
   IO_delay() ;
   outportb( GPIOBASE + GPIO5X_OFFSET , al_char ) ;

   al_char = inportb( GPIOBASE + GPIO6X_OFFSET ) | 0x02 ;  // SGPO61P[1]
   IO_delay() ;
   outportb( GPIOBASE + GPIO6X_OFFSET , al_char ) ;

   fixdelay_15us() ;

   al_char = inportb( GPIOBASE + GPIO5X_OFFSET ) & 0x54 ;  // Mask SGPI52,54,56
   al_char1= inportb( GPIOBASE + GPIO6X_OFFSET ) & 0x01 ;  // Mask SGPI60
   al_char = al_char | al_char1 ;


  if( al_char == 0x55 ) goto gpio_done ;

   al_char1 = 0x04 ; // Test4  failed.

  gpio_test_failed :
   outportb ( config_VT1211 , 0xaa ) ;   // exit  config  mode
   printf("\n\n Error#0C : PPAP-2020 GPIO test failed. \( for External loopback\)t%X=>0x%X\n", al_char1, al_char);
   exit(1) ;  //

   gpio_done :


 } // end of P2020_GPIO_TEST()


// ************

 // ********************************************************************************
 // *                                                                              *
 // *  ----------------------     MAIN   Program     -------------------------- *
 // *                                                                              *
 // ********************************************************************************
unsigned char main ()
 {

        unsigned int i,j,k;
        unsigned char al_char ;



    // *****  First :  define the Multiplexed  pins --- start
       outportb ( config_VT1211 , 0x87 ) ;    // enter  config  mode
       outportb ( config_VT1211 , 0x87 ) ;
       fixdelay_15us();

   // CR24Bit7_P1 , Define  the  multiplexed pin,pin121,  as  GPIO17
   // CR24_PFF , Define  the  multiplexed pin,pin121,  as  GPIO17~10

       outportb ( config_VT1211 , 0x24 ) ;
     //  al_char = inportb ( config_VT1211 +1 ) | 0x80 ;
       al_char = inportb ( config_VT1211 +1 ) | 0xFF ;
```

```
    outportb ( config_VT1211 , 0x24 ) ;
    outportb ( config_VT1211+1 , al_char ) ;

 // CR25Bit5_P1 , Define the multiplexed pins,pin66~97, as GPIO  pins
    outportb ( config_VT1211 , 0x25 ) ;
    al_char = inportb ( config_VT1211 +1 ) | 0x20 ;
    outportb ( config_VT1211 , 0x25 ) ;
    outportb ( config_VT1211+1 , al_char ) ;
 // *****  First :  define the Multiplexed pins  --- end

 // Select IO direction and Non-inverse
 //  GPO17,34,35,37,40,41,44,45,46,47,50,51
 //  GPI42,43
   outportb ( config_VT1211 , 0x07 ) ;
   outportb ( config_VT1211+1 , GPIO_LDN ) ; // point to LDN of GPIO

   outportb ( config_VT1211 , 0x30 ) ;
   al_char = inportb ( config_VT1211 + 1 ) | 0x01 ;
   outportb ( config_VT1211 , 0x30 ) ;
   outportb ( config_VT1211+1 , al_char ) ; // enable  GPIO

// Get GPIOBASE , High byte <-- LD8_Rx60 , Low byte <-- LD8_Rx61
   outportb ( config_VT1211 , 0x60 ) ;
   al_char = inportb ( config_VT1211 + 1 )  ;
   GPIOBASE = al_char ;
   GPIOBASE = GPIOBASE << 0x08  ;   // High byte of GPIOBASE

   outportb ( config_VT1211 , 0x61 ) ;
   al_char = inportb ( config_VT1211 + 1 )  ; // Low byte of GPIOBASE

   GPIOBASE = GPIOBASE + al_char ;    // Get the GPIOBASE


 // ---- init GPO34,35,37 direction and polarity ---
   outportb ( config_VT1211 , 0xF0 ) ;                    //Select GPIO port 3
   al_char = ( inportb ( config_VT1211+1 ) & 0xF9 ) | 0x01 ; //Bit[2,1,0]P[001]
   outportb ( config_VT1211 , 0xF0 ) ;                    //point to GPIO30~37
   outportb ( config_VT1211+1 , al_char ) ;

   outportb ( config_VT1211 , 0xF2 ) ;    // Non inverse
   outportb ( config_VT1211 +1 , 0x00 ) ;

   outportb ( config_VT1211 , 0xF1 ) ;
//    al_char =  inportb ( config_VT1211+1 )  | 0xB0 ; // 1: output direction
   al_char =  ( inportb ( config_VT1211+1 )  | 0xB5 ) & 0xFD  ; // 1: output direction
   outportb ( config_VT1211 , 0xF1 ) ;
   outportb ( config_VT1211+1 , al_char ) ;
// ------------ End of GPO34,35,37 ----------




  // ---- init GPO40,41,44~47  , GPI42,43 direction and polarity---
   outportb ( config_VT1211 , 0xF0 ) ;
   al_char = ( inportb ( config_VT1211+1 ) & 0xFA ) | 0x02 ; // Bit[2,1,0]P[010]
   outportb ( config_VT1211 , 0xF0 ) ;                    // point to GPIO40~47
   outportb ( config_VT1211+1 , al_char ) ;

   outportb ( config_VT1211 , 0xF2 ) ;                    // Non inverse
```

```
    outportb ( config_VT1211 +1 , 0x00 ) ;

    outportb ( config_VT1211 , 0xF1 ) ;
    al_char = ( inportb ( config_VT1211+1 ) & 0xF3 ) | 0xF3 ; // 1: output direction
    outportb ( config_VT1211 , 0xF1 ) ;                    // 0: Input direction
    outportb ( config_VT1211+1 , al_char ) ;
// ------------ End of GPO40,41,44~47, GPI42,43 ----------

  // ---- init GPO50,51 direction and polarity ---
    outportb ( config_VT1211 , 0xF0 ) ;
    al_char = ( inportb ( config_VT1211+1 ) & 0xFB ) | 0x03 ; // Bit[2,1,0]P[011]
    outportb ( config_VT1211 , 0xF0 ) ;                    // point to GPIO50~57
    outportb ( config_VT1211+1 , al_char ) ;

    outportb ( config_VT1211 , 0xF2 ) ;                    // Non inverse
    outportb ( config_VT1211 +1 , 0x00 ) ;

    outportb ( config_VT1211 , 0xF1 ) ;
    al_char =  inportb( config_VT1211+1 ) | 0x03 ;          //1:output direction
    outportb ( config_VT1211 , 0xF1 ) ;
    outportb ( config_VT1211+1 , al_char ) ;
// ------------ End of GPO50,51 ----------

// ---- init GPIO60,61  polarity ---------
    outportb ( config_VT1211 , 0xF0 ) ;
    al_char = ( inportb ( config_VT1211+1 ) & 0xFC ) | 0x04 ; // Bit[2,1,0]P[100]
    outportb ( config_VT1211 , 0xF0 ) ;                    // point to GPIO60~67
    outportb ( config_VT1211+1 , al_char ) ;

    outportb ( config_VT1211 , 0xF2 ) ;                    // Non inverse
    outportb ( config_VT1211 +1 , 0x00 ) ;

// ------------ End of GPIO60,61 ----------

// ---- init GPO17 direction and polarity ---
    outportb ( config_VT1211 , 0xF0 ) ;
    al_char = inportb ( config_VT1211+1 ) & 0xF8   ; // Bit[2,1,0]P[000]
    outportb ( config_VT1211 , 0xF0 ) ;            // point to GPIO10~17
    outportb ( config_VT1211+1 , al_char ) ;

    outportb ( config_VT1211 , 0xF2 ) ;            // Non inverse
    outportb ( config_VT1211 +1 , 0x00 ) ;

    outportb ( config_VT1211 , 0xF1 ) ;
    al_char =  inportb( config_VT1211+1 ) | 0xFF ;  // 1: output direction
    outportb ( config_VT1211 , 0xF1 ) ;
    outportb ( config_VT1211+1 , al_char ) ;

  //  al_char = inportb( GPIOBASE + GPIO1X_OFFSET ) & 0x7F   ; // GP17 : init to 0
    al_char = ( inportb( GPIOBASE + GPIO1X_OFFSET ) & 0x7F ) | 0x7F  ; // GP17 : init to 0 , GP16 -->1
    IO_delay() ;
    outportb( GPIOBASE + GPIO1X_OFFSET , al_char ) ; // P0  ,


// ------------ End of GPO17 ----------


Main_menu :
system("cls") ;
```

```
// printf("\n");
printf(" PORTWELL PPAP-2020 BP_WDT %s,All rights reserved.\n", VER_DATE) ;
al_char = inportb( GPIOBASE + GPIO4X_OFFSET ) & 0x08 ;// Check Bypass status , GP43 , high = bypass
if ( al_char == 0x08 )
printf ("       Current mode status: --> Bypass\(Mode LED OFF\).\n");
else printf ("       Current mode status: --> Normal\(Mode LED ON\).\n" );


// printf("      !!!!! TEST PROGRAM FOR MS-DOS ENVIRONMENT !!!!! \n") ;

printf(" For PPAP-2020 By_Pass function, BP_WDT, and SYS_WDT test. <<< DOS program >>>\n");
printf(" ***************************************************************************\n");
printf(" *                    BP_WDT       |   SYS_WDT  (ZR2 and later)      *\n");
printf(" *  Twd :  SIO_GP44..GP47, GP50,51    |   SIO_GP15..10                *\n");
printf(" *           LOADJ      : SIO_GP34    |   SIO_GP16                    *\n");
printf(" *           BP_STS_Bit : SIO_GP43    |   None                        *\n");
printf(" *           DIS_BP_WDT : SIO_GP41    |   SIO_GP30                    *\n");
printf(" *  WDT_Time_out_STS_bit : SIO_GP42    |   SIO_GP31                    *\n");
printf(" *  Reset_WDT_timeout_bit: SIO_GP35    |   SIO_GP32                    *\n");
printf(" *  SW_SET_NORMAL_MODE#_A  : SIO_GP37 |   None                          *\n");
printf(" *  SW_SET_BY_PASS_MODE#_A : SIO_GP40 |   None                          *\n");
printf(" ***************************************************************************\n");
printf(" < 1 > : ENABLE BP_WDT, REFRESH BP_WDT 1000 TIMES,& DISABLE BP_WDT \n");
printf(" < 2 > : SET NORMAL and BY_PASS mode test.\n");
printf(" < A > : BP_WDT AUTO TEST \( First <2>:SET Mode test , then <1>:BP_WDT test.\)\n");
printf(" < B > : SET TO BY_PASS AND RETURN TO MS-DOS.\n");
printf(" < C > : RUN BP EXT. LOOPBACK by connecting GPIO Port J14 to a test fixture.\n");
printf(" < D > : Test BP_WDT status bit and RESET the status bit after BP_WDT timeout.\n");
printf(" < E > : ENABLE SYS_WDT, REFRESH SYS_WDT 1000 TIMES,& DISABLE SYS_WDT \n");
printf(" < F > : Enable SYS_WDT \( This will reset system after 1 second.\)\n");
printf(" < G > : SYS_WDT AUTO Refresh TEST \( Run all sub-items in <E> \)\n");
printf(" < 3 > : EXIT.\n");
printf("         Your choice :<_>\b\b");

  i =getche() ;   printf(">\n");

// ************************************
// *                                  *
// *   Main Menu :  Selection <3>     *
// *                                  *
// ************************************
  if ( i == 0x33 )
    {
     printf("\n\n Test Item  <3> .  Return to DOS \n");
     outportb ( config_VT1211 , 0xaa ) ;   // exit  config  mode
     exit(0) ;
    }

// ************************************
// *                                  *
// *   Main Menu :  Selection <B>     *
// *                                  *
// ************************************
  else if ( (i == 0x42) || (i== 0x62) )
  {
    al_char = Set_bypass() ;
    if ( al_char == 0x00 ) goto  next_return_to_dos ;
    printf("\n\n  Error#02 :\"Set to Bypass\"  test  failed. \n");
    outportb ( config_VT1211 , 0xaa ) ;   // exit  config  mode
```

```
    exit(1) ;
   next_return_to_dos :
     printf("\n\n  Test Item  <B> .  Return to DOS \n");
     outportb ( config_VT1211 , 0xaa ) ;   // exit  config  mode
     exit(0) ;
  }   // end of selection B

// ************************************
// *                                  *
// *   Main Menu :  Selection <C>     *
// *                                  *
// ************************************
  // RUN BP EXT. LOOPBACK by connecting P2020 ZR2 GPIO & an ext. test fixture.
  else if ( (i == 0x43) || (i== 0x63) )
  {
    // VT1211 Super IO GP52~57, 60~61 are used for PPAP-2020 GPIO

    Set_bypass() ;

       P2020_GPIO_TEST() ;
       printf("\n\n 2 RJ45 Twist pairs test OK.         ^_^ Will go on the test of the rest 2 pairs.\n");
       printf("\n\n Toggle the Blue or Red switch, then press any key to continue the test...");
       getche() ;

       P2020_GPIO_TEST() ;
       printf("\n\n      This BP group test OK.          ^_^ Will go on the test of next BP group.\n");
       printf("\n\n Toggle the BLACK switch, then press any key to continue the test of next BP...");
       getche();

       P2020_GPIO_TEST() ;
       printf("\n\n 2 RJ45 Twist pairs test OK.         ^_^ Will go on the test of the rest 2 pairs.\n");
       printf("\n\n Toggle the Blue or Red switch, then press any key to continue the test...");
       getche() ;

       P2020_GPIO_TEST() ;
       printf("\n\n BYPASS EXTERNAL LOOPBACK TEST OKAY. ^_^ Press any key to return to main menu...");
       getche() ;

     goto Main_menu ;

  } // end of Selection C



// ************************************
// *                                  *
// *   Main Menu :  Selection <D>     *
// *                                  *
// ************************************
  else if (  (i == 0x44) || (i== 0x64 ) )
  {
        printf("\n BP_WDT status bit will be checked first. ");
        printf("\n Then BP_WDT enabled and expired in 1 second. ");
        printf("\n Finally , Reset the status bit to 1 \n");
        WDT_disable() ;

   // Get BP_WDT status  bit  first  GPI42 should be "1"
      al_char = inportb( GPIOBASE + GPIO4X_OFFSET ) & 0x04 ;
      if( al_char == 0x04 ) goto  go_on_test_D ;
```

```
      printf("\n Current BP_WDT status bit is \"0\" \( i.e. BP_WDT has expired \) \n") ;
      printf("\n Continue the test? \"Y\":continue ; Others : Quit and Return to DOS.") ;
      j = getche() ;
      if ( j == 'Y' || j == 'y' ) goto continue_the_test_D ;
      printf("\n\n  Error#03: BY_WDT has expired before the test. Quit and Return to DOS\n\n");
      outportb ( config_VT1211 , 0xaa ) ;   // exit  config  mode
      exit(1) ;

 continue_the_test_D :
   // Will reset BP_WDT to normal state "1"
   CLEARJ_BP_WDT_STS_HLH() ;

 go_on_test_D :


// Twd load into WDT[5:0] ( GP51,50,47,46,45,44 ) first ==========
 al_char = inportb( GPIOBASE + GPIO5X_OFFSET ) & 0xFC ;
 IO_delay() ;
 outportb( GPIOBASE + GPIO5X_OFFSET , al_char ) ; // write GP51,50
 al_char = ( inportb( GPIOBASE + GPIO4X_OFFSET ) & 0x0F ) | 0x10 ;
 IO_delay() ;
 outportb( GPIOBASE + GPIO4X_OFFSET , al_char ) ; // write GP47,46,45,44

  LOADJ_HLH() ;

  sleep(2) ; // Pause 2 seconds to wait for BP_WDT expire

  al_char = inportb( GPIOBASE + GPIO4X_OFFSET ) & 0x04 ;

  if( al_char == 0x00 ) goto  BP_WDT_status_t1 ;
   printf("\n\n  Error#04: BY_WDT status bit error. Quit and Return to DOS\n\n");
   outportb ( config_VT1211 , 0xaa ) ;   // exit  config  mode
   exit(1) ;

  BP_WDT_status_t1 :
  CLEARJ_BP_WDT_STS_HLH() ;

  al_char = inportb( GPIOBASE + GPIO4X_OFFSET ) & 0x04 ;

  if( al_char == 0x04 ) goto  BP_WDT_status_t2 ;
   printf("\n\n  Error#05: BY_WDT status bit error. Quit and Return to DOS\n\n");
   outportb ( config_VT1211 , 0xaa ) ;   // exit  config  mode
   exit(1) ;

  BP_WDT_status_t2 :
    Set_normal() ;

      printf("\n\n BP_WDT status bit test ----> PASS , ^_^ .  Press any key to continue...");
      getche() ;
      goto Main_menu ;
  }  // end of choice D


// ************************************
// *                                  *
// *   Main Menu :  Selection <A>     *
// *                                  *
// ************************************
  else if (  (i == 0x41) || (i== 0x61 ) )
```

```
{
   printf("\n\n\(Set BYPASS & NORMAL Modes\) testing.-->Sound of Relay operation and LED on/off.\n");

   for ( k=1 ; k <=5 ; k++ )
   {
     al_char = Set_bypass() ;
     if ( al_char == 0x00 ) goto  next_normal_test_auto ;
     printf("\n\n  Error#06 :\"Set to Bypass\"  test  failed. \n");
     outportb ( config_VT1211 , 0xaa ) ;   // exit  config  mode
     exit(1) ;
   next_normal_test_auto :
     WDT_disable() ;
     delay(800) ;

     al_char = Set_normal() ;
      if ( al_char == 0x00 ) goto  next_bypass_test_auto ;
     printf("\n\n  Error#07 :\"Set to Normal\"  test  failed. \n");
     outportb ( config_VT1211 , 0xaa ) ;   // exit  config  mode
     exit(1) ;
   next_bypass_test_auto :
     WDT_disable() ;
     delay(800);
    }

  printf("\n\n Start  BP_WDT  refresh  testing  ..... \n");

 Sub_menu_display( 0x55 ,'1') ;
    al_char = WDT_REFRESH_1K(1,989) ;
      if ( al_char == 0x1b ) goto Main_menu ; // ESC return
 Sub_menu_display( 0x55 ,'2') ;
    al_char = WDT_REFRESH_1K(2,1978) ;
      if ( al_char == 0x1b ) goto Main_menu ; // ESC return
 Sub_menu_display( 0x55 ,'3') ;
    al_char = WDT_REFRESH_1K(4,3956) ;
      if ( al_char == 0x1b ) goto Main_menu ; // ESC return
 Sub_menu_display( 0x55 ,'4') ;
    al_char = WDT_REFRESH_1K(8,7912) ;
      if ( al_char == 0x1b ) goto Main_menu ; // ESC return
 Sub_menu_display( 0x55 ,'5') ;
    al_char = WDT_REFRESH_1K(14,13461) ;
      if ( al_char == 0x1b ) goto Main_menu ; // ESC return
 Sub_menu_display( 0x55 ,'6') ;
    al_char = WDT_REFRESH_1K(20,19231) ;
      if ( al_char == 0x1b ) goto Main_menu ; // ESC return
 Sub_menu_display( 0x55 ,'7') ;
    al_char = WDT_REFRESH_1K(26,25000) ;
      if ( al_char == 0x1b ) goto Main_menu ; // ESC return
 Sub_menu_display( 0x55 ,'8') ;
    al_char = WDT_REFRESH_1K(30,28846) ;
      if ( al_char == 0x1b ) goto Main_menu ; // ESC return
 Sub_menu_display( 0x55 ,'9') ;
    al_char = WDT_REFRESH_1K(34,32692) ;
      if ( al_char == 0x1b ) goto Main_menu ; // ESC return
 Sub_menu_display( 0x55 ,'a') ;
    al_char = WDT_REFRESH_1K(38,36538) ;
      if ( al_char == 0x1b ) goto Main_menu ; // ESC return
 Sub_menu_display( 0x55 ,'b') ;
    al_char = WDT_REFRESH_1K(42,40385) ;
      if ( al_char == 0x1b ) goto Main_menu ; // ESC return
```

```
    Sub_menu_display( 0x55 ,'c') ;
      al_char = WDT_REFRESH_1K(46,44231) ;
        if ( al_char == 0x1b ) goto Main_menu ; // ESC return
    Sub_menu_display( 0x55 ,'d') ;
      al_char = WDT_REFRESH_1K(50,48077) ;
        if ( al_char == 0x1b ) goto Main_menu ; // ESC return
    Sub_menu_display( 0x55 ,'e') ;
      al_char = WDT_REFRESH_1K(54,51923) ;
        if ( al_char == 0x1b ) goto Main_menu ; // ESC return
    Sub_menu_display( 0x55 ,'f') ;
      al_char = WDT_REFRESH_1K(56,53846) ;
        if ( al_char == 0x1b ) goto Main_menu ; // ESC return
    Sub_menu_display( 0x55 ,'g') ;
      al_char = WDT_REFRESH_1K(58,55769) ;
        if ( al_char == 0x1b ) goto Main_menu ; // ESC return
    Sub_menu_display( 0x55 ,'h') ;
      al_char = WDT_REFRESH_1K(60,57692) ;
        if ( al_char == 0x1b ) goto Main_menu ; // ESC return
    Sub_menu_display( 0x55 ,'i') ;
      al_char = WDT_REFRESH_1K(62,59615) ;
        if ( al_char == 0x1b ) goto Main_menu ; // ESC return

   printf("\n\n    BP_WDT AUTO TEST ----> PASS , ^_^ .  Press any key to continue...");
   getche() ;
   goto Main_menu ;
  }


// ************************************
// *                                  *
// *   Main Menu :  Selection <1>     *
// *                                  *
// ************************************
  else if ( i == 0x31 ) // retrigger 100 times

  {
    Sub_Menu :

    Sub_menu_display( 0x55 ,'K') ;

    i =getche() ;   printf(">\n\n");
    if ( i == 'Q' || i == 'q' ) goto Main_menu ;
    switch(i)
    {
      case 0x31 : al_char = WDT_REFRESH_1K(1,989) ;
                  break ;
      case 0x32 : al_char = WDT_REFRESH_1K(2,1978) ;
                  break ;
      case 0x33 : al_char = WDT_REFRESH_1K(4,3956) ;
                  break ;
      case 0x34 : al_char = WDT_REFRESH_1K(8,7912) ;
                  break ;
      case 0x35 : al_char = WDT_REFRESH_1K(14,13461) ;
                  break ;
      case 0x36 : al_char = WDT_REFRESH_1K(20,19231) ;
                  break ;
       case 0x37 : al_char = WDT_REFRESH_1K(26,25000) ;
                  break ;
      case 0x38 : al_char = WDT_REFRESH_1K(30,28846) ;
```

```
                    break ;
    case 0x39 : al_char = WDT_REFRESH_1K(34,32692) ;
                    break ;
    case 0x41 : al_char = WDT_REFRESH_1K(38,36538) ;
                  break ;
     case 0x61 : al_char = WDT_REFRESH_1K(38,36538) ;
                    break ;
    case 0x42 : al_char = WDT_REFRESH_1K(42,40385) ;
                    break ;
           case 0x62 : al_char = WDT_REFRESH_1K(42,40385) ;
                    break ;
    case 0x43 : al_char = WDT_REFRESH_1K(46,44231) ;
                    break ;
     case 0x63 : al_char = WDT_REFRESH_1K(46,44231) ;
                    break ;
     case 0x44 : al_char = WDT_REFRESH_1K(50,48077) ;
                    break ;
     case 0x64 : al_char = WDT_REFRESH_1K(50,48077) ;
                    break ;
    case 0x45 : al_char = WDT_REFRESH_1K(54,51923) ;
                    break ;
     case 0x65 : al_char = WDT_REFRESH_1K(54,51923) ;
                    break ;
    case 0x46 : al_char = WDT_REFRESH_1K(56,53846) ;
                    break ;
     case 0x66 : al_char = WDT_REFRESH_1K(56,53846) ;
                    break ;
     case 0x47 : al_char = WDT_REFRESH_1K(58,55769) ;
                    break ;
     case 0x67 : al_char = WDT_REFRESH_1K(58,55769) ;
                    break ;
    case 0x48 : al_char = WDT_REFRESH_1K(60,57692) ;
                    break ;
     case 0x68 : al_char = WDT_REFRESH_1K(60,57692) ;
                    break ;
    case 0x49 : al_char = WDT_REFRESH_1K(62,59615) ;
                    break ;
    case 0x69 : al_char = WDT_REFRESH_1K(62,59615) ;
                    break ;
     default   : al_char = WDT_REFRESH_1K( 1, 1 ) ;
                  break ;

  } // end of switch

   if ( al_char == 0x1b ) goto Main_menu ; // ESC return

  printf("\n\n Refresh BP_WDT 1000 times --> PASS, ^_^ ,Press any key to do more tests...") ;
  getche();

  goto  Sub_Menu ;


 }  // end of  if (i==0x31)


// ************************************
// *                                  *
// *   Main Menu :  Selection <E>     *
// *                                  *
```

```
// ***********************************
  else if ( i == 'E' || i =='e' ) // retrigger 1000 times

  {

    CLEARJ_SYS_WDT_STS_HLH() ;

    SYSWDT_Sub_Menu :

    Sub_menu_display( 0xAA ,'K') ;

    i =getche() ;   printf(">\n\n");
    if ( i == 'Q' || i == 'q' ) goto Main_menu ;
    switch(i)
    {
      case 0x31 : al_char = WDT_RFSH_1K(1,989) ;
                  break ;
      case 0x32 : al_char = WDT_RFSH_1K(2,1978) ;
                  break ;
      case 0x33 : al_char = WDT_RFSH_1K(4,3956) ;
                  break ;
      case 0x34 : al_char = WDT_RFSH_1K(8,7912) ;
                  break ;
      case 0x35 : al_char = WDT_RFSH_1K(14,13461) ;
                  break ;
      case 0x36 : al_char = WDT_RFSH_1K(20,19231) ;
                  break ;
       case 0x37 : al_char = WDT_RFSH_1K(26,25000) ;
                  break ;
      case 0x38 : al_char = WDT_RFSH_1K(30,28846) ;
                  break ;
      case 0x39 : al_char = WDT_RFSH_1K(34,32692) ;
                  break ;
       case 0x41 : al_char = WDT_RFSH_1K(38,36538) ;
                 break ;
       case 0x61 : al_char = WDT_RFSH_1K(38,36538) ;
                  break ;
      case 0x42 : al_char = WDT_RFSH_1K(42,40385) ;
                  break ;
            case 0x62 : al_char = WDT_RFSH_1K(42,40385) ;
                  break ;
      case 0x43 : al_char = WDT_RFSH_1K(46,44231) ;
                  break ;
       case 0x63 : al_char = WDT_RFSH_1K(46,44231) ;
                  break ;
       case 0x44 : al_char = WDT_RFSH_1K(50,48077) ;
                  break ;
       case 0x64 : al_char = WDT_RFSH_1K(50,48077) ;
                  break ;
      case 0x45 : al_char = WDT_RFSH_1K(54,51923) ;
                  break ;
       case 0x65 : al_char = WDT_RFSH_1K(54,51923) ;
                  break ;
      case 0x46 : al_char = WDT_RFSH_1K(56,53846) ;
                  break ;
       case 0x66 : al_char = WDT_RFSH_1K(56,53846) ;
                  break ;
       case 0x47 : al_char = WDT_RFSH_1K(58,55769) ;
                  break ;
```

```
        case 0x67 : al_char = WDT_RFSH_1K(58,55769) ;
                    break ;
       case 0x48 : al_char = WDT_RFSH_1K(60,57692) ;
                    break ;
        case 0x68 : al_char = WDT_RFSH_1K(60,57692) ;
                    break ;
       case 0x49 : al_char = WDT_RFSH_1K(62,59615) ;
                    break ;
       case 0x69 : al_char = WDT_RFSH_1K(62,59615) ;
                    break ;
        default   : al_char = WDT_RFSH_1K( 1, 1 ) ;
                   break ;

     } // end of switch

      if ( al_char == 0x1b ) goto Main_menu ; // ESC return

     printf("\n\n Refresh SYS_WDT 1000 times --> PASS, ^_^ ,Press any key to do more tests...") ;
     getche();

     goto  SYSWDT_Sub_Menu ;


    }  // end of  if (i=='e')


// ************************************
// *                                  *
// *   Main Menu :  Selection <F>     *
// *                                  *
// ************************************
  else if ( i == 'F' || i =='f' ) // Enable System WDT and Let WDT expire.( will reset system )
   {
 // System WDT Twd load into SWDT[5:0] ( GP15~10 ) first ==============
 al_char = ( inportb( GPIOBASE + GPIO1X_OFFSET ) & 0xC1 ) | 0x41 ;
 IO_delay() ;
 outportb( GPIOBASE + GPIO1X_OFFSET , al_char ) ; // write GP15~10

  SYSWDT_LOADJ_HLH() ;
 //    LOADJ_HLH() ; // for  testing  only

    printf("\n\n This test will reset system after 1 second. \n" );
    // otherwise  test  failed
    sleep(3) ;
    printf("\n\n  ERROR#0A : Test  failed  of  system  WDT enabling. \n") ;
    outportb ( config_VT1211 , 0xaa ) ;   // exit  config  mode
    exit(1) ;

   } // end of if(i=='f')

// ************************************
// *                                  *
// *   Main Menu :  Selection <G>     *
// *                                  *
// ************************************
  else if ( i == 'G' || i =='g' ) // AUTO test  system WDT
  {
      CLEARJ_SYS_WDT_STS_HLH() ;
```

```
  printf("\n\n Start  SYS_WDT  refresh  testing  ..... \n");

 Sub_menu_display( 0xAA ,'1') ;
   al_char = WDT_RFSH_1K(1,989) ;
     if ( al_char == 0x1b ) goto Main_menu ; // ESC return
 Sub_menu_display( 0xAA ,'2') ;
   al_char = WDT_RFSH_1K(2,1978) ;
     if ( al_char == 0x1b ) goto Main_menu ; // ESC return
 Sub_menu_display( 0xAA ,'3') ;
   al_char = WDT_RFSH_1K(4,3956) ;
     if ( al_char == 0x1b ) goto Main_menu ; // ESC return
 Sub_menu_display( 0xAA ,'4') ;
   al_char = WDT_RFSH_1K(8,7912) ;
     if ( al_char == 0x1b ) goto Main_menu ; // ESC return
 Sub_menu_display( 0xAA ,'5') ;
   al_char = WDT_RFSH_1K(14,13461) ;
     if ( al_char == 0x1b ) goto Main_menu ; // ESC return
 Sub_menu_display( 0xAA ,'6') ;
   al_char = WDT_RFSH_1K(20,19231) ;
     if ( al_char == 0x1b ) goto Main_menu ; // ESC return
 Sub_menu_display( 0xAA ,'7') ;
   al_char = WDT_RFSH_1K(26,25000) ;
     if ( al_char == 0x1b ) goto Main_menu ; // ESC return
 Sub_menu_display( 0xAA ,'8') ;
   al_char = WDT_RFSH_1K(30,28846) ;
     if ( al_char == 0x1b ) goto Main_menu ; // ESC return
 Sub_menu_display( 0xAA ,'9') ;
   al_char = WDT_RFSH_1K(34,32692) ;
     if ( al_char == 0x1b ) goto Main_menu ; // ESC return
 Sub_menu_display( 0xAA ,'a') ;
   al_char = WDT_RFSH_1K(38,36538) ;
     if ( al_char == 0x1b ) goto Main_menu ; // ESC return
 Sub_menu_display( 0xAA ,'b') ;
   al_char = WDT_RFSH_1K(42,40385) ;
     if ( al_char == 0x1b ) goto Main_menu ; // ESC return
 Sub_menu_display( 0xAA ,'c') ;
   al_char = WDT_RFSH_1K(46,44231) ;
     if ( al_char == 0x1b ) goto Main_menu ; // ESC return
 Sub_menu_display( 0xAA ,'d') ;
   al_char = WDT_RFSH_1K(50,48077) ;
     if ( al_char == 0x1b ) goto Main_menu ; // ESC return
 Sub_menu_display( 0xAA ,'e') ;
   al_char = WDT_RFSH_1K(54,51923) ;
     if ( al_char == 0x1b ) goto Main_menu ; // ESC return
 Sub_menu_display( 0xAA ,'f') ;
   al_char = WDT_RFSH_1K(56,53846) ;
     if ( al_char == 0x1b ) goto Main_menu ; // ESC return
 Sub_menu_display( 0xAA ,'g') ;
   al_char = WDT_RFSH_1K(58,55769) ;
     if ( al_char == 0x1b ) goto Main_menu ; // ESC return
 Sub_menu_display( 0xAA ,'h') ;
   al_char = WDT_RFSH_1K(60,57692) ;
     if ( al_char == 0x1b ) goto Main_menu ; // ESC return
 Sub_menu_display( 0xAA ,'i') ;
   al_char = WDT_RFSH_1K(62,59615) ;
     if ( al_char == 0x1b ) goto Main_menu ; // ESC return

 printf("\n\n    SYS_WDT AUTO TEST ----> PASS , ^_^ .  Press any key to continue...");
 getche() ;
```

```
   goto Main_menu ;

  } // End of if(i=='g')


// ************************************
// *                                  *
// *   Main Menu :  Selection <2>     *
// *                                  *
// ************************************
  else if ( i == 0x32 )
    {
      printf("\n\n\(Set BYPASS & NORMAL Modes\) testing.-->Sound of Relay operation and LED on/off.\n");
      for ( k=1 ; k <=5 ; k++ )
      {
        al_char = Set_bypass() ;
        if ( al_char == 0x00 ) goto  next_normal_test ;
        printf("\n\n  Error#08 :\"Set to Bypass\"  test  failed. \n");
        outportb ( config_VT1211 , 0xaa ) ;   // exit  config  mode
        exit(1) ;
      next_normal_test :
        WDT_disable() ;
        delay(800) ;

        al_char = Set_normal() ;
         if ( al_char == 0x00 ) goto  next_bypass_test ;
        printf("\n\n  Error#09 :\"Set to Normal\"  test  failed. \n");
        outportb ( config_VT1211 , 0xaa ) ;   // exit  config  mode
        exit(1) ;
      next_bypass_test :
        WDT_disable() ;
        delay(800);

      }

      printf("\n\n \(Set BYPASS & NORMAL Modes\) test OKAY ^_^ , Press any key for more tests...");

      getche() ;


      goto  Main_menu ;

    }  // end of  if (i==0x32)

    else  goto  Main_menu ;


}    //  end of Main
```

Chapter 3 Operation Guide

3.1 Brief Guide of PPAP-2020

PPAP-2020 is a Communication Appliance computing board based on VIA VT8237R+ chipset technology. PPAP-2020 has four/five on-board LAN ports to serve communication appliances, such as Firewall, which needs four Ethernet ports to connect external network (internet), demilitarized zone and internal network. Different I/O management policies can be applied respectively to individual network to achieve the highest security level. The target market segment is communication appliance including Virtual Private Network, Load Balancing, Quality of Service, Intrusion Detection, Virus Detection, Firewall and Voice Over IP.

This PPAP-2020 system board is eligible with VIA Eden processor EBGA package (Eden Esp8000) and On-board 256Mb or higher DDRAM. The enhanced on-board PCI IDE interface supports 1 drive up to PIO mode 4 timing and Ultra DMA/100 synchronous mode feature. The on-board super I/O chipset integrates two serial ports driven by two high performance 16550C-compatible UARTs to provide 16-byte send/receive FIFOs. Besides, the two Universal Serial Bus ports provide high-speed data communication between peripherals and PC.

The on-board flash ROM is used to make the BIOS update easier. The high precision Real Time Clock/Calendar is built to support Y2K for accurate scheduling and storing configuration information. All of these features make PPAP-2020 excellent in stand-alone applications.

If any of these items is damaged or missing, please contact your vendor and save all packing materials for future replacement and maintenance.

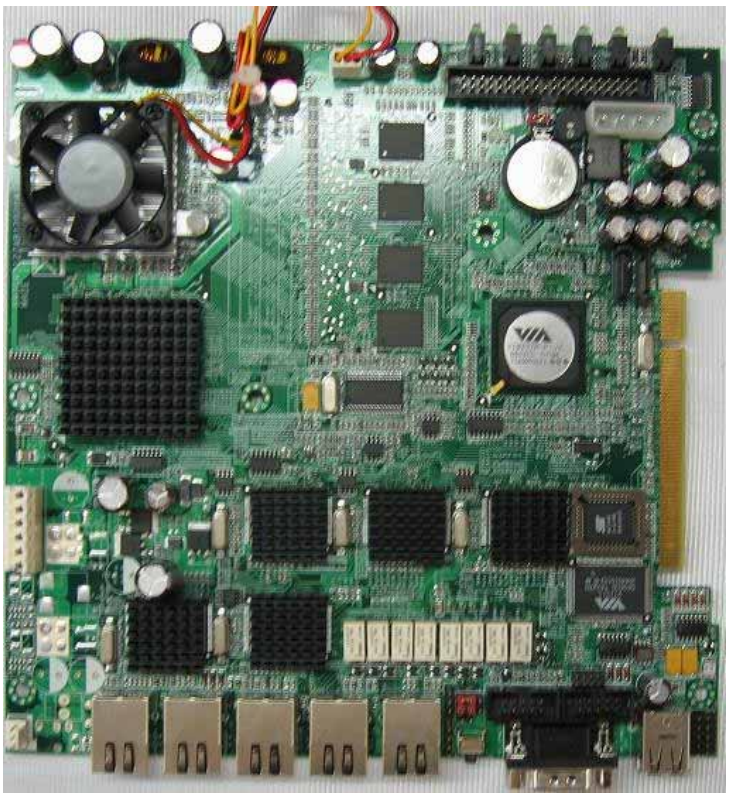


Figure 3-1 PPAP-2020 Board

3.2 System Architecture

The following illustration of block diagram will show you how PPAP-2020 gives you a highly integrated system solution. The most up-to-date system architecture of PPAP-2011 includes two main chips. It contains VIA CN700 and VIA VT823R+ to support VIA C7 processor, DDR2 SODIMM, USB 2.0 port, communication, Ultra DMA/100 IDE Master and SATA storage. The on-board super VIA VT8237R+ supports two UARTs and hardware monitoring.

PPAP-2020 has built-in onboard VIA C7 processor EBGA package (Eden C7) 533 or 400MHz system bus) for cost-effective and high performance application.

The VIA CN700 provides a completely integrated solution for the system controller and data path components in a VIA processor system. It provides optimized 64-bit DDR2 interface.

The VIA VT8237R+ provides a highly integrated multifunction for the best industry applications. It supports up to for Ultra ATA/33/66/100 IDE master interface, Universal Serial Bus **(USB2.0)** controllers,Full duplex high performance 150MB/s Dual Channel SATA interface.

All detailed operating relations are shown in ***Fig. 3-2*** (PPAP-2020 System Block Diagram).

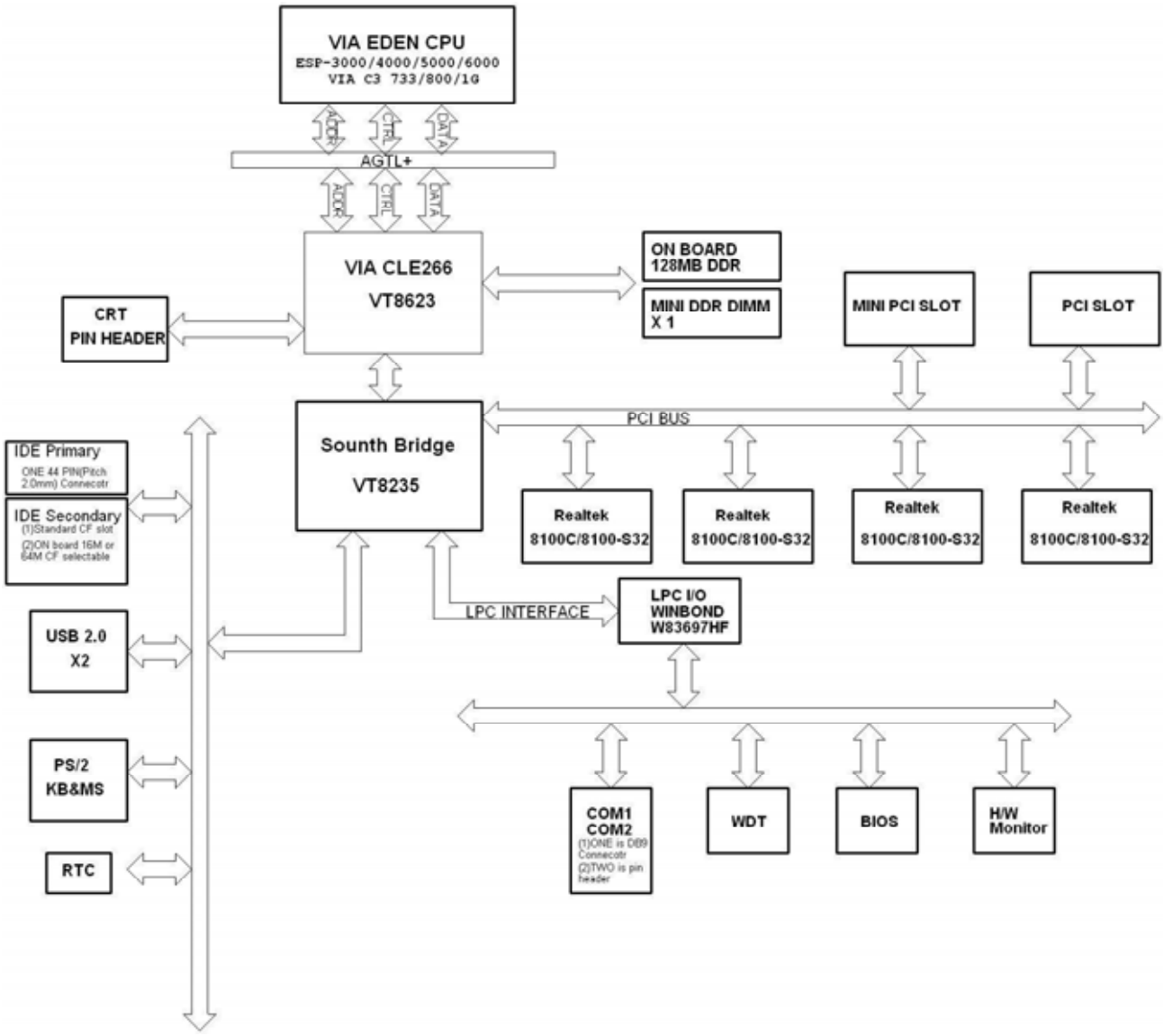


Figure 3-2 PPAP-2020 Block Diagram